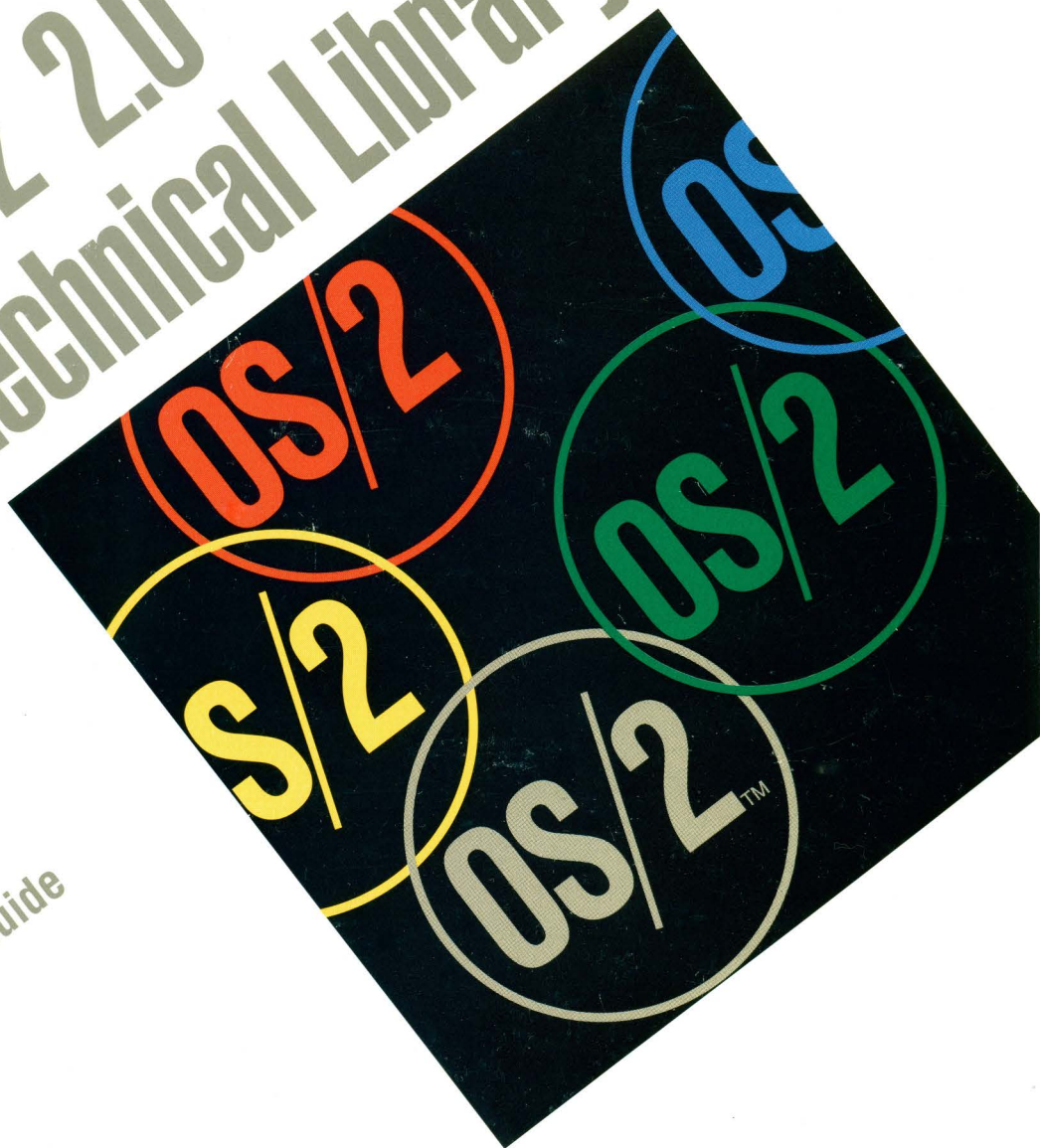
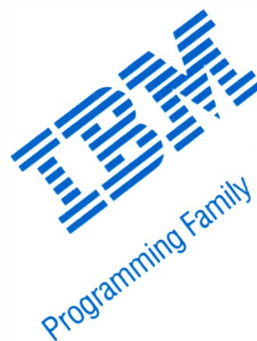


OS/2 2.0 Technical Library



Programming Guide
Volume III

Version 2.00



OS/2 2.0 Technical Library

**Programming Guide
Volume III**

Version 2.00



Programming Family

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xxi.

First Edition (March 1992)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "© (your company name) (year) All Rights Reserved."

© Copyright International Business Machines Corporation 1992. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xxi
Trademarks and Service Marks	xxi
Double-Byte Character Set (DBCS)	xxi
About This Book	xxiii
Structure of the Books	xxiv
Prerequisite Knowledge	xxiv

Presentation Manager Graphics Programming Interface

Chapter 1. Introduction to the Graphics Programming Interface	1-1
About the GPI	1-1
Presentation Spaces and Device Contexts	1-2
Graphics Primitives	1-3
Primitive Attributes	1-3
Line and Arc Primitives	1-3
Area Primitives	1-3
Polygon Primitives	1-3
Character String Primitives and Fonts	1-4
Marker Primitives	1-4
Color and Mix Attributes	1-4
Bit Maps	1-4
Metafiles	1-5
Paths	1-5
Regions	1-5
Retained Graphics	1-5
Correlation	1-6
Clipping and Boundary Determination	1-6
Coordinate Spaces and Transformation Operations and Functions	1-6
Print Job Submission and Manipulation	1-7
Program Execution	1-7
Summary	1-8
Chapter 2. Presentation Spaces and Device Contexts	2-1
About Presentation Spaces	2-1
Micro Presentation Spaces	2-1
Standard Micro Presentation Space	2-2
Cached Micro Presentation Space	2-2
Normal Presentation Spaces	2-4
Modal Graphic Systems	2-4
Current Position	2-5
Primitive Attributes	2-5
Using Presentation Spaces	2-6
Obtaining and Creating a Presentation Space	2-6
Deleting a Presentation Space	2-6
Saving and Restoring a Presentation Space	2-6
Presentation Spaces Review	2-6
About Device Contexts	2-7
Cached Device Contexts	2-7
Window Device Contexts	2-7
Normal Device Contexts	2-8

Using Device Contexts	2-9
Creating a Device Context	2-9
Associating Presentation Spaces with Device Contexts	2-10
Closing a Device Context	2-12
Determining Device Capabilities	2-12
Summary	2-13
Chapter 3. Line and Arc Primitives	3-1
About Line and Arc Primitives	3-1
Attributes of Line and Arc Primitives	3-2
Line Width and Geometric Width	3-3
Line Types	3-4
Line Color and Mix Attributes	3-5
Line Primitive Family	3-6
Lines	3-7
Polylines	3-7
Boxes	3-8
Attributes of Arc Primitives	3-10
Simple-Arc Primitive Family	3-12
Full Arcs	3-13
Partial Arcs	3-15
3-Point Arcs	3-16
Multiple-Arc Primitive Family	3-17
Fillet	3-17
Splines	3-19
Using Line and Arc Primitives	3-20
Drawing a Straight Line	3-20
Creating a Rubber-Banding Effect with a Straight Line	3-20
Drawing a Circle	3-22
Drawing an Ellipse	3-23
Drawing a Pie Slice	3-23
Drawing a Fillet	3-25
Drawing a Spline	3-26
Summary	3-27
Chapter 4. Marker Primitives	4-1
About Marker Primitives	4-1
Attributes of Marker Primitives	4-1
Marker Symbols	4-2
Marker Box	4-3
Marker Set	4-4
Marker Color and Mix Attributes	4-4
Using Marker Primitives	4-6
Drawing Marker Primitives	4-6
Selecting a New Marker	4-7
Selecting a New Marker Set	4-8
Changing the Marker Color	4-9
Summary	4-10
Chapter 5. Area and Polygon Primitives	5-1
About Area Primitives	5-1
Attributes of Area Primitives	5-1
Pattern Symbol Attribute	5-2
Pattern Reference Point Attribute	5-3
Pattern Set Attribute	5-4
Default Pattern Set	5-4

Customizing Pattern Sets	5-4
Area Colors and Mix Attributes	5-5
Area Brackets	5-7
Area Bracket Attributes	5-9
Area Boundaries	5-9
Area Construction	5-10
Alternate Mode	5-10
Winding Mode	5-11
Attribute Currentness	5-12
About Polygon Primitives	5-13
Polygon Boundaries	5-14
Polygon Construction	5-14
Polygon Overlap	5-15
Using Area and Polygon Primitives	5-15
Drawing a Single, Closed Figure	5-15
Drawing Multiple, Intersecting, Closed Figures	5-16
Creating a Custom Fill Pattern from a Bit Map	5-16
Creating a Custom Fill Pattern from a Font Character	5-19
Summary	5-20
Chapter 6. Character String Primitives	6-1
About Character String Primitives	6-1
Attributes of Character String Primitives	6-2
Character Set	6-3
Character Mode	6-3
Character Cell	6-4
Default Character Cell	6-7
Coding a Character Cell	6-7
Character Angle	6-8
Character Shear	6-10
Character Direction	6-12
Character Text Alignment	6-13
Horizontal Alignment of a Character String	6-13
Vertical Alignment of a Character String	6-14
Character Extra and Break Extra	6-15
Character Color and Mix	6-16
Using Character String Primitives	6-18
Drawing Text	6-18
Formatting Text	6-19
Positioning Text in World Coordinates	6-20
Summary	6-22
Chapter 7. Color and Mix Attributes	7-1
About Color and Mix Attributes	7-1
Color Implementation	7-1
RGB Color Encoding	7-2
Color Tables	7-3
Logical Color Table	7-3
Default Logical Color Table	7-4
Device-Independent Color Indexing	7-5
Defining a Logical Color Table	7-5
Color Tables in Index Mode	7-6
Color Tables in RGB Mode	7-6
Querying the Available Colors	7-7
Physical Color Table	7-7
Palette Manager	7-8

Realizing a Color Palette	7-8
Color Attribute	7-9
Primitive Foreground	7-9
Primitive Background	7-9
Changing the Foreground and Background Colors of Primitives	7-10
Color Output and Mix Attributes	7-11
Mix Attribute	7-11
Overpaint Mix Attribute	7-14
OR Mix Attribute	7-14
Exclusive-OR (XOR) Mix Attribute	7-15
Leave-Alone Mix Attribute	7-16
Specifying Foreground and Background Mix Attributes	7-16
Color on Advanced Display Devices	7-17
Dithering	7-17
Considerations When Using Monochrome Displays	7-17
Drawing Color Graphics on Monochrome Devices	7-18
Drawing Color Area Fill Patterns on Monochrome Devices	7-18
Using Color and Mix Attributes	7-20
Creating a Logical Color Table	7-20
Determining the Color-Table Format and Index Values	7-21
Determining the Index Value of an RGB Value	7-22
Setting the Primitive Color Attributes	7-22
Creating a Palette	7-23
Summary	7-24
Chapter 8. Bit Maps	8-1
About Bit Maps	8-1
System Implementation	8-2
Bit Map Functions	8-3
Creating a Bit Map	8-3
Creating and Loading a Custom Bit Map	8-6
Storing Color Information in a Bit Map	8-6
Drawing Bit Maps	8-8
Transferring Bit Maps	8-10
Changing the Size of the Bit Map	8-12
Specifying Mix Values	8-13
Converting between Monochrome and Color Data Bit Maps	8-15
Manipulating Single Pels	8-15
Copying Images from a Display into a Bit Map	8-15
Saving a Bit Map	8-16
Deleting a Bit Map	8-17
Making Bit Maps Available to Other Processes	8-17
Using Bit Maps	8-17
Copying an Image from a Display Screen to a Bit Map	8-17
Scaling and Drawing a Bit-Map Image	8-20
Creating a Custom Fill Pattern	8-20
Loading a Bit Map from a File	8-21
Storing a Bit Map in a Metafile	8-22
Summary	8-24
Chapter 9. Fonts	9-1
About Fonts	9-1
Image Font and Outline Font Implementation	9-2
PM-Supplied Fonts	9-2
Availability of Additional Fonts	9-4
Font Data Structures and Attributes	9-4

Glyphs, Code Pages, and Code Points	9-8
Proportional and Monospace Fonts	9-9
Kerning	9-9
FATTRS Data Structure	9-11
Record Length	9-11
Selection Indicators	9-11
Match Value	9-12
Face Name	9-12
Registry Identifier	9-12
Code Page	9-12
Maximum Baseline Extent	9-12
Average Character Width	9-13
Type Indicator	9-13
Font-Use Indicators	9-13
Public and Private Fonts	9-14
Drawing a Character String Primitive	9-14
Font Files and Dynamic-Link Libraries	9-15
Using Fonts	9-17
Selecting a Font	9-17
Explicit Font Selection	9-17
Reassociating the Presentation Space	9-21
Font Resolution	9-22
Closest-Matching Font Selection	9-22
Selecting the New Current Font	9-24
Deleting Logical Fonts and Unloading Physical Fonts	9-24
Creating Fonts	9-25
Creating Marker Sets and Pattern Sets	9-25
Using Bit Maps as Area-Fill Patterns	9-25
Using Paths with Fill Patterns	9-26
Summary	9-26
 Chapter 10. Paths	 10-1
About Paths	10-1
Path Attributes	10-2
Line Width and Geometric Width for Paths	10-3
Line End	10-4
Line Join	10-5
Cosmetic Line Attributes for Paths	10-6
Area Attributes for Paths	10-6
Path Color and Mix Attributes	10-6
Path Brackets	10-8
Path Operations	10-9
Path Outline	10-10
Path Fill	10-11
Paths in Alternate Mode	10-11
Paths in Winding Mode	10-12
Path Modification	10-13
Path-Stroking	10-14
Path Conversion to Clip Path	10-15
Path Conversion to Region	10-16
Using Paths	10-16
Drawing a Geometric (Wide) Line	10-17
Drawing Filled Polygons	10-17
Drawing Outline Text	10-18
Creating a Triangular Clip Path	10-19
Summary	10-20

Chapter 11. Regions	11-1
About Regions	11-1
System Implementation	11-2
Region Attributes	11-2
Region Creation	11-3
Region Operations	11-4
Creating Regions	11-4
Moving a Region	11-6
Determining Region Characteristics	11-6
Converting a Region to a Clip Region	11-7
Deleting a Region	11-9
Using Regions	11-9
Creating and Deleting a Region	11-9
Combining Regions	11-10
Comparing Regions	11-11
Offsetting a Region	11-11
Painting a Region	11-12
Locating a Point with Respect to a Region	11-12
Determining Coordinates of Rectangles in a Region	11-13
Summary	11-15
 Chapter 12. Creating and Drawing Retained Graphics	12-1
About Creating and Drawing Retained Graphics	12-1
Drawing Modes	12-2
Draw Mode	12-2
Retain Mode	12-2
Draw-and-Retain Mode	12-3
Creating a Graphics Segment	12-3
Filling a Graphics Segment	12-4
Closing a Graphics Segment	12-4
Segment Attributes	12-4
Chained Attribute	12-5
Fast-Chained Attribute	12-5
Actual Drawing Mode	12-6
Drawing a Retained Graphic	12-6
Segment Priority	12-8
GpiDrawSegment	12-8
GpiDrawFrom	12-8
Attribute Modes	12-9
Reusing the Presentation Space	12-10
GpiSavePS	12-10
GpiRestorePS	12-10
GpiResetPS	12-11
GpiSetPS	12-11
Nonretained Graphic Segments	12-11
Using Segment Creating and Drawing Functions	12-12
Creating a Chained Segment	12-12
Creating a Called Segment	12-13
Drawing a Segment Chain	12-13
Summary	12-14
 Chapter 13. Editing Retained Graphics and Graphics Segments	13-1
About Editing Retained Graphics and Graphics Segments	13-1
Graphics Orders	13-2
Graphics Elements	13-3
Adding Elements to a New Segment	13-4

Element Types	13-4
The Element Pointer	13-5
Labels	13-5
Graphics Segments	13-6
Initial Segment Attributes	13-6
The Dynamic Attribute	13-7
The Detectability Attribute	13-8
The Propagate-Detectability Attribute	13-8
The Visibility Attribute	13-8
The Propagate-Visibility Attribute	13-8
Changing the Attributes of a Segment	13-8
Editing a Segment	13-9
Deleting a Graphics Element	13-11
Deleting Graphics Segments	13-11
Copying a Single Graphics Element	13-11
Copying Multiple Graphics Elements	13-12
Drawing Retained Graphics	13-13
Drawing Dynamic Segments	13-14
The Drawing Controls	13-14
Using Segment Editing Functions	13-16
Summary	13-18
 Chapter 14. Correlation	14-1
About Correlation	14-1
Correlating Nonretained Graphics	14-2
Correlating Retained Graphics	14-3
Tagging Primitives within a Segment	14-4
Correlation Functions for Retained Graphics	14-4
The Correlation Input Parameters	14-5
Pick Aperture	14-10
Using Correlation	14-11
Summary	14-12
 Chapter 15. Metafiles	15-1
About Metafiles	15-1
Contents of a Metafile	15-2
Metafile Content Restrictions	15-3
Metafile Functions	15-6
Creating a Metafile	15-7
Storing Pictures in a Metafile	15-8
Playing a Metafile	15-10
Saving a Metafile	15-16
Loading a Metafile	15-16
Editing a Metafile	15-16
Copying a Metafile	15-16
Deleting a Metafile	15-17
Metafiles and the System Clipboard	15-17
Using Metafiles	15-17
Creating and Drawing into a Metafile	15-17
Drawing into a Metafile in Retain Mode	15-19
Copying a Metafile to Disk	15-20
Playing a Metafile	15-21
Summary	15-23
 Chapter 16. Clipping and Boundary Determination	16-1
About Clipping	16-1

Types of Clipping Areas	16-2
The Clip Path	16-3
The Viewing Window	16-5
The Graphics Field	16-5
The Current Clipping Region	16-6
How Clipping Is Implemented	16-8
Redrawing Nondynamic Graphics	16-8
About Boundary Determination	16-9
Using Clipping and Boundary Determination	16-11
Creating a Clip Path	16-11
Creating a Clip Region	16-12
Excluding a Rectangular Area from a Clip Region	16-12
Adding a Rectangular Area to a Clip Region	16-13
Setting the Clip Region to a Region Intersection	16-14
Determining the Size of a Clipping Area	16-14
Summary	16-15
 Chapter 17. Coordinate Spaces and Transformations	17-1
About Coordinate Spaces	17-1
World Coordinate Space	17-2
Model Space	17-3
Page Space	17-3
Device Space	17-4
Media Space	17-4
About Transformations	17-4
Identity Transformation	17-6
Moving through Coordinate Spaces Using the Identity Transformation	17-6
Applying Transformations Other Than the Identity Transformation	17-7
Combining Transformations Between a Coordinate Space Pair	17-10
Transformation Mathematics	17-11
Model for Building the Transformation Matrix	17-12
MATRIXLF, the Transformation Matrix	17-13
MATRIXLF Structure	17-13
MAKEFIXED Macro	17-14
About Transformation Operations	17-14
Scaling and Reflection Transformations	17-15
Scaling a Graphics Object	17-16
Reflecting a Graphics Object	17-16
MATRIXLF Structure for Scaling and Reflecting	17-17
Rotation Transformations	17-17
Rotating a Graphics Object	17-18
MATRIXLF Structure for Rotating	17-19
Translation Transformations	17-19
Translating a Graphics Object	17-20
MATRIXLF Structure for Translating	17-20
Shearing Transformations	17-21
Shearing a Graphics Object	17-21
MATRIXLF Structure for Shearing	17-21
About Transformation Functions	17-22
Current Transformation	17-22
Accumulating Transformations	17-22
Concatenating Transformations	17-24
Hard Coding Values for a Concatenated Matrix	17-25
Multiplying Matrix Values	17-25
Transformation Helper Functions	17-26

Applying Transformation Operations Directly to the Transformation Matrix and Using Transformation Functions	17-27
Round-Off Error	17-27
World Space-to-Model Space Transformations	17-27
Model Transformations	17-28
Segment Transformations	17-29
Instance Transformations	17-29
World Space-to-Model Space Transformation Summary	17-30
Model Space-to-Page Space Transformations	17-30
Viewing Transformations	17-30
Defining the Viewing Window	17-31
Graphics Field	17-32
Default Viewing Transformations	17-32
Page Space-to-Device Space Transformations	17-33
Presentation Pages	17-34
Page Viewports	17-34
Mapping the Presentation Page to the Device	17-34
Coding the Device Transformation	17-36
Device-Transformation Matrix	17-37
Windowing-System Transformation	17-37
Transforming Bit-Map Data	17-37
Using Coordinate Spaces and Transformations	17-39
Setting Drawing Units	17-39
Translating, Rotating, and Scaling a Picture	17-39
Shearing a Picture	17-41
Using World to Model Space Transformations	17-41
Viewing Transformation	17-42
Summary	17-44
 Chapter 18. Print Job Submission and Manipulation	18-1
About the Print Subsystem Components	18-1
Spooler	18-1
Print Job Formats	18-2
Disabling the Spooler	18-3
Print Subsystem User Interface	18-3
Queue Driver	18-4
Printer Driver	18-4
File System	18-4
Kernel Device Driver	18-4
Print Subsystem Configuration	18-4
Printing Data Flow	18-6
Submitting a Non-Presentation Manager (Base) Print Job	18-9
Submitting a Queued Presentation Manager Print Job	18-9
Page Setup Dialog	18-10
Forms Selection	18-11
Printer Setup Dialog	18-13
Job Properties Considerations	18-14
Retrieving Job Properties	18-14
Displaying Job Properties Dialog	18-15
Font Dialog and Device Font Considerations	18-17
Print Dialog and Print Processing	18-18
Creating a New Thread	18-19
Opening a Queued Device Context	18-19
Associating a Presentation Space	18-21
Starting a Print Job	18-22
Querying the Device Capabilities	18-23

Formatting the Page	18-23
Starting the Next Page	18-23
Ending the Print Job	18-23
Disassociating the Presentation Space	18-25
Closing the Device Context	18-25
Print-to-File Considerations	18-25
Network Printing Considerations	18-26
Drag/Drop Protocol Considerations	18-26
PMPRINT/PMPLLOT Queue Processor Parameters	18-27
Submitting a Direct Presentation Manager Print Job	18-30
Submitting a Print Job Directly to the Spooler	18-30
Spooler Management and Configuration	18-31
Print Job Management	18-31
Summary	18-32

Appendixes

Appendix A. Matrix Multiplication	A-1
Appendix B. GPI Functions	B-1
Appendix C. Graphics Attributes	C-1
Appendix D. Graphics Orders	D-1
The Graphics-Orders Header File (PMORD.H)	D-1
Decoding Graphics Orders	D-3
Naming Conventions	D-4
Index	X-1

Figures

1-1.	The Flow of an Application's Graphics Commands	1-2
2-1.	Line Drawn with the GpiLine Function.	2-4
2-2.	DEVOPENSTRUC Structure	2-9
2-3.	Creating a Printer Device Context	2-9
2-4.	Creating a Metafile Device Context	2-10
2-5.	Associating, Disassociating, and Reassociating a Presentation Space	2-11
2-6.	Associating a Cached Device Context with a Normal Presentation Space	2-11
2-7.	Creating a Normal Presentation Space	2-12
3-1.	Sample Pie Chart Created with Line and Arc Primitives	3-1
3-2.	Sample Bar Graph Created with Line and Arc Primitives	3-1
3-3.	Sample Blueprint Created with Line and Arc Primitives	3-2
3-4.	The Nine Line Types	3-4
3-5.	Line and Arc Primitives	3-5
3-6.	GpiLine	3-7
3-7.	GpiLine Used to Draw a Single Point	3-7
3-8.	GpiPolyLine	3-8
3-9.	GpiPolylineDisjoint	3-8
3-10.	The Box	3-9
3-11.	Quarter-Circle Box-Corner Rounding	3-9
3-12.	Box with Rounded Corners	3-10
3-13.	The ARCPARAMS Values in World Coordinates	3-11
3-14.	The Ellipse	3-13
3-15.	Tilted Ellipse	3-14
3-16.	The Full Arc	3-15
3-17.	The Partial Arc	3-16
3-18.	The 3-Point Arc	3-16
3-19.	The Fillet	3-17
3-20.	Fillet with Sharpness Specified	3-18
3-21.	The Spline	3-19
3-22.	Spline with No Discontinuity of Gradient	3-19
3-23.	Drawing a Straight Line	3-20
3-24.	Rubber-Banding a Straight Line	3-21
3-25.	Drawing a Circle	3-22
3-26.	Drawing an Ellipse	3-23
3-27.	Closed Figure Bounded by Chord and Arc	3-24
3-28.	Drawing a Pie Slice	3-24
3-29.	Drawing a Fillet	3-25
3-30.	Drawing a Hyperbolic Curve	3-25
3-31.	Drawing a Spline	3-26
4-1.	Marker Primitives	4-1
4-2.	The Base Marker Set	4-3
4-3.	Marker Primitives	4-5
4-4.	Drawing a Graph	4-7
4-5.	Selecting a New Marker	4-7
4-6.	Selecting a New Marker Set	4-8
4-7.	Changing the Marker Color	4-9
5-1.	An Area	5-1
5-2.	The Base Pattern Symbol set.	5-3
5-3.	The Pattern Reference Point	5-3
5-4.	Area Primitives	5-5
5-5.	Defining an Area Primitive	5-8

5-6.	Calculating Filled Areas Constructed in Alternate Mode	5-10
5-7.	The Box	5-11
5-8.	Area Constructed in Different Directions in Winding Mode	5-12
5-9.	Area Constructed in the Same Direction in Winding Mode	5-12
5-10.	Drawing a Closed Figure	5-15
5-11.	Drawing Multiple Intersecting Closed Figures	5-16
5-12.	Creating a Custom Fill Pattern from a Bit Map	5-17
5-13.	Creating a Custom Fill Pattern from a Font Character	5-19
6-1.	Image and Outline Fonts	6-2
6-2.	Character Cell Measurements	6-5
6-3.	Effect of the Character Cell on an Outline Font	6-6
6-4.	Effect of the Character Cell on an Image Font in CM_MODE1	6-6
6-5.	Effect of the Character Cell on an Image Font in CM_MODE2	6-7
6-6.	Effect of the Character Angle on an Outline Font	6-8
6-7.	Effect of the Character Angle on an Image Font in CM_MODE1	6-9
6-8.	Effect of the Character Angle on an Image Font in CM_MODE2	6-9
6-9.	Effect of Character Shear on an Outline Font	6-10
6-10.	Effect of Character Shear on an Image Font in CM_MODE1	6-11
6-11.	Effect of Character Shear on an Image Font in CM_MODE2	6-11
6-12.	Effect of X- and Y-Values on Character Shear	6-12
6-13.	Effect of Character Direction on an Image Font or Outline Font	6-12
6-14.	Horizontal Positioning of Text Strings	6-13
6-15.	Vertical Positioning of Text Strings	6-14
6-16.	The Cumulative Effect of Break Values	6-15
6-17.	Character String Primitives	6-16
6-18.	Drawing Text	6-18
6-19.	Horizontal Positioning of Text Strings	6-21
7-1.	Logical Color Table	7-3
7-2.	Foreground of a Primitive	7-9
7-3.	Background of a Primitive	7-10
7-4.	Overpaint Foreground Mix Attribute	7-14
7-5.	Overpaint Background Mix Attribute	7-14
7-6.	OR Mix Attribute	7-15
7-7.	Exclusive-OR (XOR) Mix Attribute	7-16
7-8.	Leave-Alone Background Mix Attribute	7-16
7-9.	Creating a Logical Color Table	7-21
7-10.	Determining Color-Table Format and Index Values	7-21
7-11.	Determining the Index Value of an RGB Value	7-22
7-12.	Setting Primitive Color Attributes	7-22
7-13.	Setting Foreground Color Attributes	7-23
7-14.	Creating a Palette	7-23
8-1.	Bits and Pels in a Bit-Map Image	8-1
8-2.	Bit Map Shown on Two Types of Displays	8-2
8-3.	Creating and Displaying Bit Maps	8-5
8-4.	A Bit Map and Its Associated Color Table	8-8
8-5.	Image-Primitive Bits and Pels	8-10
8-6.	Points Count for Bit-Block Transfers	8-12
8-7.	Copying a Display-Screen Image to a Bit Map	8-18
8-8.	Scaling and Drawing a Bit-Map Image	8-20
8-9.	Creating a Custom Fill Pattern	8-21
8-10.	Loading a Bit Map from a File	8-22
8-11.	Storing a Bit Map in a Metafile	8-23
9-1.	Image and Outline Characters	9-2
9-2.	Some Font Metrics	9-6
9-3.	a-Space, b-Space, and c-Space	9-10
9-4.	The Character String Primitive	9-15

9-5.	Creating a Custom Font	9-15
9-6.	Module Definition File for Custom Font File	9-16
9-7.	Linking a Custom Font File	9-16
9-8.	Resource for Custom Font File	9-16
9-9.	Creating the Resource File for a Custom Font File	9-16
9-10.	Determining the Number of Public Fonts Loaded	9-18
9-11.	Selecting a Font	9-19
9-12.	Selecting an Image Font	9-21
9-13.	Using Clip Paths to Create Characters	9-26
10-1.	Drawing Techniques Achieved with Paths	10-1
10-2.	Geometric Lines and Normal Lines	10-4
10-3.	Closing Unattached Geometric Lines	10-5
10-4.	Joining Wide Lines	10-5
10-5.	Path Objects	10-7
10-6.	Alternate and Winding Fill Modes	10-11
10-7.	Calculating Filled Paths Constructed in Alternate Mode	10-12
10-8.	Paths Constructed in the Same Direction in Winding Mode	10-12
10-9.	Paths Constructed in Different Directions in Winding Mode	10-13
10-10.	Defining Lines with a Geometric Width	10-14
10-11.	Triangular Clip Path	10-16
10-12.	Drawing a Wide Line	10-17
10-13.	Drawing Filled Polygons	10-17
10-14.	Drawing Outline Text	10-18
10-15.	Creating a Triangular Clip Path	10-19
11-1.	Defining a Region	11-2
11-2.	Combining Overlapping Regions	11-5
11-3.	Combining Disjoint Regions	11-6
11-4.	Repairing the Screen with Clip Regions	11-8
11-5.	Creating and Deleting a Region	11-10
11-6.	Combining Regions	11-10
11-7.	Comparing Regions	11-11
11-8.	Offsetting a Region	11-11
11-9.	Painting a Region	11-12
11-10.	Locating a Point in a Region	11-12
11-11.	Determining Rectangle Coordinates in a Region	11-14
12-1.	Graphics Segments in a Presentation Space	12-3
12-2.	Chained and Called Segments	12-7
12-3.	Chained and Called Segments Reordered Using GpiSetSegmentPriority	12-8
12-4.	A LIFO Stack with Four Items	12-10
12-5.	Creating a Chained Graphics Segment	12-12
12-6.	Creating a Called Graphics Segment	12-13
12-7.	Drawing a Segment Chain	12-13
13-1.	Structure of a Graphics Segment	13-1
13-2.	Graphics Orders	13-3
13-3.	Inserting a New Element in a Segment	13-10
13-4.	Replacing an Element with a New Element	13-10
13-5.	Editing the Contents of a Graphics Segment	13-17
14-1.	The Pick Aperture	14-1
14-2.	Correlating Nonretained Graphics	14-3
14-3.	Retained Segment Correlation with One Hit	14-6
14-4.	Retained Segment Correlation with Multiple Hits	14-7
14-5.	alSegTag Data Structure	14-8
14-6.	Multiple Hits from a Called Segment	14-9
14-7.	alSegTag Data Structure for Different IMaxDepth Values	14-10
14-8.	Performing a Correlation Operation	14-11

15-1.	Metafile Functions	15-7
15-2.	Creating a Metafile	15-18
15-3.	Copying a Graphic Segment to a Metafile	15-19
15-4.	Copying a Metafile to Disk	15-20
15-5.	Playing a Metafile	15-21
15-6.	Playing a Metafile Using Font, Color, and Fill-Pattern Descriptions	15-21
15-7.	Playing a Metafile Using Transformation and Page Units	15-22
16-1.	Triangular Clip Path	16-1
16-2.	Inclusive/Inclusive and Inclusive/Exclusive Clipping	16-3
16-3.	The Clipping Path	16-4
16-4.	The Viewing Window	16-5
16-5.	The Graphics Field	16-6
16-6.	Screen Repairing	16-7
16-7.	A Hexagon and Circle	16-8
16-8.	Defining a Clipping Region	16-9
16-9.	The Bounding Rectangle	16-10
16-10.	Creating a Clip Path	16-11
16-11.	Creating a Clip Region	16-12
16-12.	Excluding a Rectangular Area from a Clip Region	16-12
16-13.	Adding a Rectangular Area to a Clip Region	16-13
16-14.	Setting the Clip Region to a Region Intersection	16-14
16-15.	Determining the Size of a Clipping Area	16-14
17-1.	Viewing Pipeline	17-5
17-2.	Picture Assembly Process	17-8
17-3.	Different Spaces	17-9
17-4.	Different Spaces with an Exploded View	17-10
17-5.	Determining a FIXED Equivalent	17-14
17-6.	Flag Before Transformation	17-15
17-7.	Scaling by 0.5	17-16
17-8.	Reflection	17-17
17-9.	Rotation Counterclockwise through 90°	17-19
17-10.	Translation by (8,5)	17-20
17-11.	Shearing Along the X-Axis	17-21
17-12.	Translating before Scaling	17-23
17-13.	Scaling before Translating	17-24
17-14.	Segments	17-28
17-15.	Presentation-Page Space	17-31
17-16.	Scrolling the Presentation Page	17-33
17-17.	Mapping a Picture from the Presentation Page to the Device	17-35
17-18.	Device Space	17-36
17-19.	Setting Drawing Units	17-39
17-20.	Translating, Rotating, and Scaling a Triangle	17-40
17-21.	Shearing a Picture	17-41
17-22.	Using World-to-Model Transformations	17-41
17-23.	Using Viewing Transformations	17-43
18-1.	Example Configurations of Queues, Devices, and Printer Objects	18-5
18-2.	Overview of the Application Interface and Data Flow	18-7
18-3.	Application Page Setup Dialog	18-10
18-4.	Querying the Printer Using DevQueryHardcopyCaps	18-12
18-5.	Application Printer Setup Dialog	18-13
18-6.	Displaying the Job Properties Dialog	18-16
18-7.	Character Clipping Results for Device and System Fonts	18-17
18-8.	Application Printer Setup Dialog	18-18
18-9.	Reassociating Presentation Space with Device Contexts	18-21
18-10.	An Effect of Programming in Pels	18-22
18-11.	Flow for Device Escapes	18-24

18-12. Examples Using the ARE and FIT Parameters	18-29
--	-------

Tables

1-1.	Types of Coordinate Spaces	1-7
1-2.	GPI Functions	1-8
2-1.	Functions that Create Standard Micro Presentation Spaces	2-2
2-2.	Functions that Obtain Cached Micro Presentation Spaces	2-3
2-3.	Functions that Create Normal Presentation Spaces	2-4
2-4.	Presentation Space Features and Restrictions	2-6
2-5.	Normal Device Contexts	2-8
2-6.	Presentation Space and Device Context Functions	2-13
2-7.	Presentation Space and Device Context Structures	2-14
3-1.	Line Attribute Default Values	3-2
3-2.	Line-Width Attributes as Interpreted by PM	3-3
3-3.	Standard Line Types	3-4
3-4.	Functions that Draw Straight Lines	3-6
3-5.	Direction of the Arc	3-12
3-6.	Functions that Draw Simple Arcs	3-12
3-7.	Functions that Draw Multiple-Arcs	3-17
3-8.	Fillet Sharpness Values	3-18
3-9.	Line and Arc Primitive Functions	3-27
3-10.	Line and Arc Primitive Structures	3-27
4-1.	Marker Attribute Default Values	4-2
4-2.	The Base Marker Set	4-2
4-3.	Marker Functions	4-10
4-4.	Marker Structures	4-10
5-1.	Area Attribute Default Values	5-2
5-2.	The Base Pattern Set	5-2
5-3.	Area-Primitive Functions	5-20
5-4.	Area-Primitive Structure	5-20
6-1.	Character String Attribute Default Values	6-2
6-2.	Character Mode Effects on Character Attributes	6-4
6-3.	Horizontal Alignment Values	6-14
6-4.	Vertical Alignment Values	6-14
6-5.	Break Values and Their Effects	6-15
6-6.	World Coordinate Values	6-20
6-7.	Functions for Determining Width of Character String Primitives	6-20
6-8.	Character String Primitive Functions	6-22
6-9.	Character String Primitive Structures	6-23
7-1.	Eight Standard Colors	7-2
7-2.	RGB Values	7-3
7-3.	Default Logical Color Table	7-4
7-4.	Device Independent Color Indexes	7-5
7-5.	Foreground Mix Attributes	7-12
7-6.	Background Mix Attributes	7-13
7-7.	Color and Mix Attribute Functions	7-24
7-8.	Color and Mix Attribute Structures	7-25
8-1.	Standard Bit-Map Formats	8-7
8-2.	Bit-Map Drawing Functions	8-9
8-3.	Bit-map Data Compression Rules	8-12
8-4.	Possible Settings for the Index Bits	8-14
8-5.	Mix Value Names	8-14
8-6.	Bit-Map Functions	8-24
8-7.	Bit-Map Structures	8-25
9-1.	Examples of Line Weight and Font Appearance Categories	9-1

9-2.	Available Image Fonts	9-3
9-3.	Available Outline Fonts	9-3
9-4.	Font Metric Attributes	9-4
9-5.	Filling in the FATTRS Structure	9-23
9-6.	Font-Use Indicator Considerations	9-23
9-7.	Font Functions	9-26
9-8.	Font Structures	9-27
10-1.	Comparison of Paths and Areas	10-1
10-2.	Path Attribute Default Values	10-2
10-3.	Standard Geometric Line End Types	10-4
10-4.	Standard Geometric Line Join Types	10-5
10-5.	Functional Sequence for Drawing Outline Text and Figures	10-10
10-6.	Path Functions	10-20
10-7.	Path Structures	10-20
11-1.	Region Functions	11-15
11-2.	Region Structures	11-15
12-1.	The Current Drawing Mode	12-6
12-2.	Retained Graphics and Segment Functions	12-14
13-1.	Graphics Orders	13-2
13-2.	Segment Graphics Drawing Modes	13-3
13-3.	Graphic Segment Attributes and Initial Settings	13-6
13-4.	Drawing Controls	13-15
13-5.	Editing Graphics and Segment Functions	13-18
14-1.	Correlation Functions	14-12
15-1.	Drawing Mode Dependencies When Recording Metafiles	15-8
15-2.	Drawing Mode Dependencies When Playing Metafiles	15-10
15-3.	GpiPlayMetaFile Options	15-11
15-4.	PMF_RESET Options for GpiPlayMetaFile	15-12
15-5.	PMF_SUPPRESS Options for GpiPlayMetafile	15-13
15-6.	PMF_LOADTYPE Options for GpiPlayMetafile	15-14
15-7.	PMF_LCIDS Options for GpiPlayMetafile	15-15
15-8.	PMF_COLORTABLES Options for GpiPlayMetafile	15-15
15-9.	PMF_COLORREALIZABLE Options for GpiPlayMetafile	15-15
15-10.	PMF_DEFAULTS Options for GpiPlayMetafile	15-16
15-11.	Metafile Functions	15-23
16-1.	Clipping Areas and Coordinate Space Summary	16-2
16-2.	Clipping Functions	16-15
16-3.	Clipping Structure	16-15
17-1.	Single-Coordinate vs. Multi-Coordinate Space Systems	17-2
17-2.	World Coordinate to Model Space Transformation Combinations	17-11
17-3.	Model to Page Space Transformation Combinations	17-11
17-4.	Transformation Factors in the MATRIXLF Data Structure.	17-13
17-5.	Transformations	17-14
17-6.	World to Model Space Transformation Summary	17-30
17-7.	Coordinate-Space and Transformation Functions	17-44
17-8.	Coordinate-Space and Transformation Structure	17-45
18-1.	Common Form Sizes	18-11
18-2.	Printing and Spooler Functions	18-32
18-3.	Printing and Spooler Structures	18-33
B-1.	Where GPI Functions Can Be Called	B-2
B-2.	The Current Drawing Mode	B-9

Notices

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM	IBM C/2
Operating System/2	OS/2
Systems Application Architecture	SAA
Personal System/2	PS/2
Common User Access	CUA
Presentation Manager	XGA
LaserPrinter	ProPrinter

The following terms, denoted by a double-asterisk (**) in this publication, are trademarks of other corporations, as follows:

Helvetica	Trademark of the Linotype Company
Times New Roman	Trademark of The Monotype Corporation Public Limited Company
Adobe	Trademark of Adobe Systems Incorporated
PostScript	Trademark of Adobe Systems Incorporated
Microsoft	Trademark of Microsoft Corporation
Windows	Trademark of Microsoft Corporation
HP	Trademark of Hewlett-Packard Company
LaserJet	Trademark of Hewlett-Packard Company
SEIKO	Trademark of SEIKO Epson Corporation
Epson	Trademark of SEIKO Epson Corporation

Double-Byte Character Set (DBCS)

Throughout this publication, you will see reference to specific values for character strings. The values are for single-byte character set (SBCS). If you use the double-byte character set (DBCS), notice that one DBCS character equals two SBCS characters.

About This Book

The three volumes of the *IBM OS/2 2.0 Programming Guide* provide information and code examples to enable you to start writing source code using the functions in the application programming interface (API) of the OS/2 2.0 operating system. Each volume covers a different facet of the operating system, as follows:

Programming Guide: Volume I—Control Program Programming Interface

Introduces you to the Control Program Programming Interface and describes the functionality provided by the base operating system.

Programming Guide: Volume II—Presentation Manager Window Programming Interface

Describes the Presentation Manager* (PM) Window Programming Interface. This volume will familiarize you with the windowed, message-based PM user interface.

Programming Guide: Volume III—Graphics Programming Interface (*this book*)

Describes the Graphics Programming Interface. This volume provides information on how to prepare graphical output for display and printing.

For complete and comprehensive information about the API, refer to the *OS/2 2.0 Control Program Programming Reference* and the *Presentation Manager Programming Reference—Volumes I, II, and III*.

For information on how to compile and link your programs, refer to the compiler publications for the programming language you are using.

The OS/2 2.0 operating system is a 32-bit system, and this guide is about programming 32-bit applications. (Sixteen-bit applications still are supported by the operating system however.)

To illustrate programming with the API, this guide makes extensive use of code fragments. Also, there are sample applications available with the *Developer's Toolkit for OS/2 2.0* (Toolkit). You should familiarize yourself with the operation of each sample from a user's viewpoint. That will help you understand the code in the samples.

Structure of the Books

Most chapters of these books are divided into two sections: *about* the topic and *using* the functions related to that topic. The first section of each chapter provides concepts, terms, and background material; the second section describes the applicable functions and is divided into subsections, each providing information about how to accomplish a specific task. Code fragments are included for most of the functions.

Prerequisite Knowledge

These books are for application designers and programmers who are familiar with the following:

- Information contained in the *Application Design Guide*
- Information contained in the Control Program and Presentation Manager reference materials
- C Programming Language.

Note: Programming experience on a multitasking operating system also would be helpful.

Presentation Manager Graphics Programming Interface

Chapter 1. Introduction to the Graphics Programming Interface

The OS/2[®] 2.0 Presentation Manager[®] (PM) Graphics Programming Interface (GPI) provides more than 200 functions. These functions enable a user to create, display, and print graphic images in a PC-based environment, using a variety of techniques, media, and formats.

The major advantage of the GPI is device-independence, which is achieved through the linking of presentation spaces with device contexts, and the use of a wide selection of graphics primitives.

This chapter describes the parts of the operating system that enable you to write PM applications, using presentation space-device context associations to circumvent having to communicate directly with device drivers. An overview of the following topics is presented:

- Presentation spaces and device contexts
- Graphics primitives
 - Line and arc primitives
 - Area primitives and polygons
 - Character string primitives and fonts
 - Marker primitives
- Color and mix attributes
- Bit maps and metafiles
- Paths and regions
- Retained and nonretained graphics
- Correlation
- Clipping and boundary determination
- Coordinate spaces and transformation operations and functions
- Print job submission and manipulation.

Subsequent chapters offer in-depth descriptions of the GPI—its components, capabilities, and advantages—to provide the information you need to write applications that can prepare graphical output for display and printing.

About the GPI

In a PM application, you use the graphics functions (most of which are prefixed *Gpi*) to program the desired graphics output. Many of the GPI functions are described in this book. For a complete alphabetic listing of the GPI functions, along with their syntax and parameters, refer to the *Presentation Manager Programming Reference*. Virtually all of the GPI calls conform to the IBM Systems Application Architecture[®] (SAA[®]).

The applications you write probably will receive window messages about the images to be created; your task, then, is to write how to process this input into output. Refer to the *OS/2 Programming Guide, Volume II*, for details about the window messages, the visual parts of the PM user interface, and the programming functions for creating and using windows, so that you can produce window-based graphics applications.

Presentation Spaces and Device Contexts

A *presentation space* is a data structure—the programming equivalent of a blank piece of paper on which graphic images are created before being sent to an output device. An output device can be, for example, a printer or plotter, memory bit map, display screen, or even a facsimile card. A PM application must direct all drawing to a presentation space.

The graphics engine and device drivers are responsible for getting output from a presentation space to an output or hardcopy device. The mapping from the presentation space to the hardcopy device is handled by a device context that is associated (linked) with the presentation space.

A *device context* also is a data structure—its purpose is to translate graphics commands made to its associated presentation space into commands that the physical device can convert to displayed information. Figure 1-1 illustrates how an application's graphics commands flow through a presentation space, to a device context, and on to the physical device.

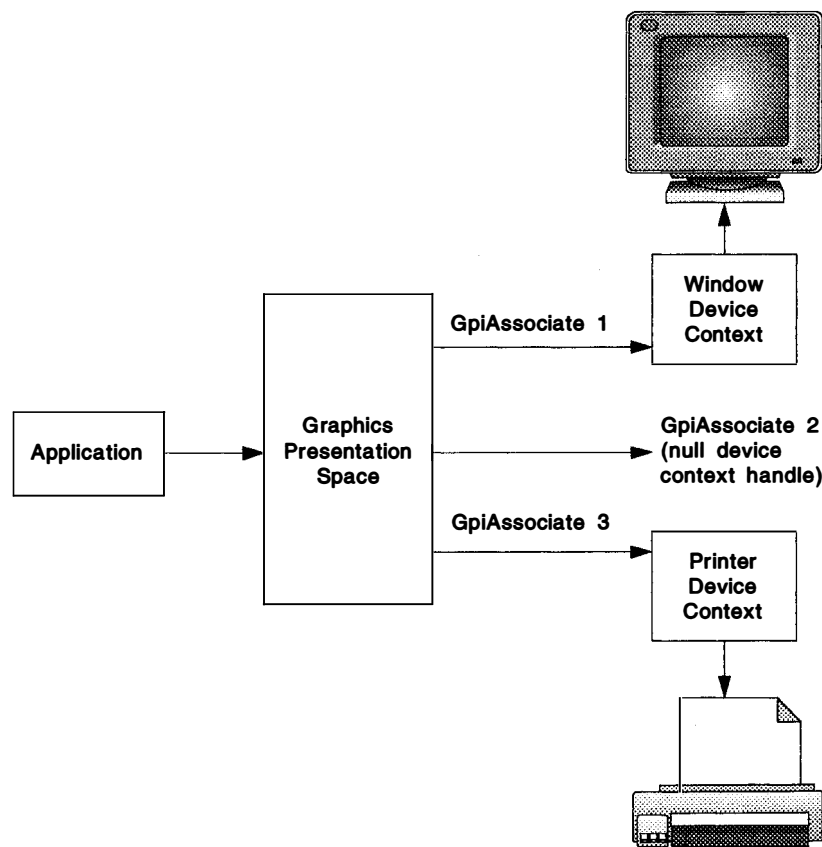


Figure 1-1. The Flow of an Application's Graphics Commands

Graphics Primitives

Graphics primitives are building blocks provided by the PM programming interface from which to construct pictures. There are several types of primitives:

- Line and arc
- Area and polygon
- Character string
- Marker.

Primitive Attributes

Each graphic primitive has a set of properties that further defines its appearance. A line, for example, can be drawn in different colors and different widths; it can be dotted, solid, or even invisible. Such properties are called *primitive attributes*.

In general, primitive attributes are set for the *type* of primitive rather than for a single instance of the primitive. If you set the line-type attribute to `LINETYPE_DOT`, for example, all lines subsequently drawn appear as dotted lines.

All primitive attributes have default settings that apply when a presentation space is first created and continue to apply until you change them. When an attribute value is set, it is called the *current attribute value*, and remains in effect until changed.

Line and Arc Primitives

Simple drawing applications, such as pie and bar charts and graphs, use line and arc primitives as their drawing tools. Computer-aided design (CAD) applications combine the numerous styles and forms of straight and curved lines that compose line and arc primitives to draw more complex pictures.

Line and arc primitive attributes are stored in the `LINEBUNDLE` data structure. Their relative attributes include width, type, color, and mix.

See Chapter 3, “Line and Arc Primitives” for detailed information about line and arc primitives and their use in PM applications.

Area Primitives

An *area* is a closed shape that can be defined with line and arc primitives; *area primitives* are produced by functions that can define and draw an area.

Area primitives are contained in the `AREABUNDLE` data structure. Area primitive attributes include pattern symbol and reference point, pattern set, foreground and background colors, and foreground and background mix modes.

Areas are also referred to as *area brackets* because the functions that create and define an area always are *bracketed* by the functions `GpiBeginArea` and `GpiEndArea`. The power of area brackets is their complete flexibility to draw any combination of curved or straight lines and combine the results into a closed figure.

Polygon Primitives

A *polygon* is a closed plane figure bounded by straight lines. A *polygon primitive* is any set of polygons with specified vertices that can be filled or filled and outlined.

The GPI has a function (`GpiPolygons`) that allows you to draw multiple straight-lined closed areas outside of an area bracket; but like areas, they can be adjacent, intersecting, or completely separate.

See Chapter 5, “Area and Polygon Primitives” for guidance on how and when to use area primitives and polygons in PM applications.

Character String Primitives and Fonts

A *character string primitive* is used to create text. A *font* is a collection of character string primitives that share a common height, line weight, and appearance.

Both character string primitive functions and font functions are used to ascertain the available fonts, select a font for text output, draw a string of the selected characters, and modify a character string.

Character string primitives are stored in the CHARBUNDLE data structure. Attributes of the character string primitives include set, precision, cell, angle, direction, text alignment, color, and mix.

See Chapter 6, “Character String Primitives” for details about character string primitives, and Chapter 9, “Fonts” for details about fonts, and how to use both in PM applications.

Marker Primitives

Marker primitives are graphics objects, used, for example, to indicate the peaks and valleys on a line graph. The GPI has a base marker set that includes visible marker symbols, such as squares, diamonds, stars, circles, and crosses.

Marker primitives are contained in the MARKERBUNDLE data structure. Marker primitive attributes consist of marker symbols, marker box, the aforementioned marker set, colors, and mix modes.

See Chapter 4, “Marker Primitives” for detailed information about marker primitives and how to use them in PM applications.

Color and Mix Attributes

Color and *Mix* are two attributes that apply to primitives. The color attribute describes the color of a line, arc, character, area, or image. The mix attribute controls how each primitive is combined with an existing drawing and affects, among other things, the resulting color when primitives of different colors overlap.

Some primitives have foreground and background color attributes. For example, the character primitive has a foreground color attribute that specifies the color of the character and a background color attribute that specifies the color surrounding the character. Primitives that have foreground and background color attributes also have foreground and background mix attributes.

See Chapter 7, “Color and Mix Attributes” for details about color implementation using the GPI.

Bit Maps

A *bit map* is a graphic image stored in memory. Bit maps can be created using the Icon Editor or hard-coded using GPI functions. The bit maps can be stored in a memory device context or in disk files. PM supports both monochrome and color bit maps.

Applications can use bit maps for the following:

- To store and display scanned images, icons, and pointers
- To create fill patterns for area primitives and paths.
- To animate graphics for multimedia displays.

See Chapter 8, "Bit Maps" for more information about bit maps, icons, and pointers, and how to make them available to the PM-application user.

Metafiles

Metafiles are files that contain low-level graphics commands that describe a picture.

The OS/2 operating system supports long-term retention of graphic so that pictures can be saved for use by any PM application, or even by non-PM applications that support the Mixed Object Document Content Architecture (MO:DCA) interchange standard. Graphics data that is to be exchanged among PM and non-PM applications is transferred to those applications in a metafile.

See Chapter 15, "Metafiles" for complete details about metafiles and their use in both PM and non-PM applications.

Paths

A *path* is an object defined by a sequence of graphic primitive and attribute functions issued between GpiBeginPath and GpiEndPath. A path can be used for defining a filled area, a complex clipping shape, an outline, and for drawing geometric (wide) lines. As a matter of fact, paths are the only means of drawing lines with a geometric (*scalable*) width, and clipping to circular, elliptical, or other nonrectangular figures, as well as being an efficient way of producing text in an outline font.

See Chapter 10, "Paths" to learn how to define and use paths with several different drawing techniques.

Regions

Regions are rectangular graphics objects in an application's device space that can be used to clip output, draw one or more discrete filled rectangles, or draw a filled polygon consisting of intersecting rectangles. Regions also are used to repaint the client area of a window or part of an object in a picture.

Regions are device-dependent and are defined in device coordinates. Each region is created for a device that is currently associated with a presentation space. The system implements regions as part of a device context.

See Chapter 11, "Regions" for more information on how to use regions in PM applications.

Retained Graphics

There are two types of graphics output in the OS/2 operating system: retained and nonretained. *Retained graphics* is a presentation space model in which an entire graphics picture is stored for subsequent display and editing.

An application draws nonretained graphics output using a graphics function. The output appears immediately on the output device, such as a window. If part of the picture is erased or has to be repeated, the application must issue the same function as many times as necessary; this is the primary drawback of nonretained graphics.

Retained graphics provide many advantages such as convenience, flexibility, editing capability, more powerful and useful graphics, and better organization. A possible disadvantage of retained graphics is the additional memory required to store the data.

PM applications store graphics in *segments*. Primarily, a graphics segment is a means of grouping and storing graphics primitives and their attributes. Although usually graphics segments are related in some way, they do not have to be.

See Chapter 12, "Creating and Drawing Retained Graphics" for detailed information on how to create and replay retained graphics. See Chapter 13, "Editing Retained Graphics and Graphics Segments" for information on how to edit retained graphics and graphics segments.

Correlation

Correlation is the process of determining which primitives or primitive group in a picture are at a given position on the display. Correlation, most commonly, is used in graphics applications; for instance, to identify a primitive or primitive group selected by the user. It should be noted, however, that correlation also can be used in non-graphic applications; for example, if your application models the operation of a calculator and allows a user to select numbers for mathematical operations.

See Chapter 14, "Correlation" for low-level information about the correlation process.

Clipping and Boundary Determination

Clipping is a process by which all output outside the defined clip boundary of a picture is discarded (clipped), and only the parts inside the clip boundary (*clipping area*) are displayed.

Most often, clipping boundaries are defined by the application. The PM interface performs some clipping automatically when, for example, the application's graphic output is clipped to fit a client area (user *work area*) or the device output area for a hard-copy device.

Boundary determination is an operation to compute the size of the smallest rectangle that encloses a graphics object on the screen, for the purpose of updating only the affected parts of the screen when that graphics object is moved or changed.

See Chapter 16, "Clipping and Boundary Determination" for comprehensive information about clipping and boundary determination for PM applications.

Coordinate Spaces and Transformation Operations and Functions

Transformations (also called *transforms*) enable an application to control the size and orientation of graphic output on any output device. The GPI transform functions provide for scaling, translation, rotation, and shear in both graphics pictures and text. Almost all of these GPI functions draw their output in a conceptual area called a *world coordinate space*.

The following table describes the five kinds of coordinate spaces used in GPI transformations:

Table 1-1. Types of Coordinate Spaces	
Name	Description
World coordinate space	A space where an application specifies its coordinates for drawing.
Model space	A conceptual space in which a picture is constructed by the output of the model transform matrix.
Default page coordinate space	A space representing a picture defined in page coordinates without any scaling or scrolling.
Page coordinate space	A conceptual space in which a picture is constructed by the output of the default viewing transform matrix.
Device coordinate space	The coordinate space of a device.

See Chapter 17, "Coordinate Spaces and Transformations" for information on all the available transforms and functions for GPI graphics transformations.

Print Job Submission and Manipulation

Printing is the process of producing hardcopy output. An application can submit the job using the *spooler*, which will direct the job to the appropriate printer. Alternatively, an application can submit the job *directly* to the printer. Direct submission can halt all user input to an application until printing is complete.

If you choose to use the spooler, you can *manipulate the job* while it is in a spool queue. A common example of manipulation is increasing the number of copies to be printed or deleting the job entirely.

See Chapter 18, "Print Job Submission and Manipulation" for more information on how to print jobs or manipulate them once inside a queue.

Program Execution

Graphics programming is influenced by the underlying execution of the operating system. While the *OS/2 2.0 Programming Guide, Volume I*, will familiarize you with the concepts of operating system execution, following are some of the concepts that you will be concerned with when programming for the GPI:

A *session* is one or more processes with its own virtual console. A virtual console is a virtual screen—either a character-based full screen or a PM window—and buffers for keyboard and mouse input.

A *process* is the application's code, data, and other resources—such as file handles, semaphores, pipes, queues, and so forth—in memory. The operating system considers every application it loads to be a process.

A *thread* is a dispatchable unit of execution that consists of a set of instructions, related CPU register values, and a stack. Every process has at least one thread and can have many threads running at the same time. The application runs when the operating system turns over control to a thread in the process. A thread is the basic unit of execution scheduling.

Graphics output can be slow, so it is wise to perform lengthy drawing operations on an asynchronous thread. As a result, the main thread is free to process its message queue and remain responsive to user input. User input goes to only the main thread, so good communication between the main thread and the drawing thread is very important if you are to achieve rapid response to user requests.

For example, if a user changes or deletes graphics, the main thread must be able to interrupt the processing of the asynchronous thread to pass on the revised instructions. The main thread issues `GpiSetStopDraw`, with the value `SDW_ON`—this suspends all drawing to the graphics presentation space named in `GpiSetStopDraw` and establishes the *stop draw* condition.

Drawing to *other* presentation spaces in the same process and in different processes is unaffected by this stop-draw condition. But no drawing can begin from another thread to the named presentation space while the stop-draw condition is in effect. Any drawing operations to the same presentation space that are in progress on other threads of the same process when you issue this function are terminated with a warning. When the main thread receives an acknowledgment from the drawing thread, it can clear the suspension by issuing `GpiSetStopDraw` with the value `SDW_OFF`.

Summary

Following is a summary of some of the GPI's frequently-used functions.

Table 1-2 (Page 1 of 4). GPI Functions	
Attribute Functions	
GpiMove	Moves the current position to the specified point.
GpiQueryAttrs	Returns current attributes for the specified primitive type.
GpiQueryCharSet	Returns the character-set local identifier (ICID) as set by <code>GpiSetCharSet</code> .
GpiQueryCurrentPosition	Returns the value of current position.
GpiQueryLineType	Returns the current cosmetic line-type attribute as set by <code>GpiSetLineType</code> .
GpiQueryLineWidth	Returns the current value of the cosmetic line-width attribute as set by <code>GpiSetLineWidth</code> .
GpiQueryMix	Returns the current value of the (character) foreground color-mixing mode as set by <code>GpiSetMix</code> .
GpiQueryPatternSet	Returns the current value of the pattern-set identifier as set by <code>GpiSetPatternSet</code> .
GpiSetArcParams	Sets the current arc parameters.
GpiSetAttrs	Sets the attributes for the specified primitive type.
GpiSetCharBox	Sets the current character-box attribute to the specified value.
GpiSetCurrentPosition	Sets the current position to the specified point.
GpiSetLineType	Sets the current cosmetic line-type attribute.
GpiSetLineWidth	Sets the current cosmetic line-width attribute.

Table 1-2 (Page 2 of 4). GPI Functions

GpiSetMix	Sets the current background mix attribute for each individual primitive type.
GpiSetPattern	Sets the current value of the pattern-symbol attribute.
Bit Map Functions	
GpiCreateBitmap	Creates a bit map and returns the bit-map handle.
GpiDeleteBitmap	Deletes a bit map.
GpiQueryBitmapBits	Transfer data from a bit map to application storage.
GpiQueryBitmapHandle	Returns the handle of the bit map currently tagged with the specified local identifier.
GpiQueryBitmapParameters	Returns information about a bit map identified by the bit-map handle.
GpiQueryDeviceBitmapFormats	Returns the formats of bit maps supported internally by the device driver.
GpiSetBitmap	Sets a bit map as the currently selected bit map in a memory device context.
GpiSetBitmapid	Tags a bit map with a local identifier so that it can be used as a pattern set containing a single member.
Device Functions	
DevCloseDC	Closes a device context.
DevEscape	Enables applications to access facilities of a device not otherwise available through the API.
DevOpenDC	Creates a device context.
DevPostDeviceModes	Returns and, optionally, sets job properties.
DevQueryCaps	Queries the device characteristics.
DevQueryDeviceNames	Queries the presentation driver to return the names, descriptions, and data types of the devices it support.
DevQueryHardcopyCaps	Queries the hard-copy capabilities of a device.
Drawing Functions	
GpiBeginArea	Begins the construction of an area.
GpiBeginPath	Specifies the start of a path.
GpiBitBit	Copies a rectangle of bit-map image data.
GpiCharString	Draws a character string starting at the current position.
GpiEndArea	Ends construction of a shaded area.
GpiEndPath	Ends the specification of a path started by GpiBeginPath.
GpiErase	Clears the output display of the device context associated with the specified presentation space to the reset color.
GpiIntersectClipRectangle	Sets the new clip region to the intersection of the current clip region and the specified rectangle.

Table 1-2 (Page 3 of 4). GPI Functions

GpiLine	Draws a straight line from the current position to the specified end point.
GpiPaintRegion	Paints a region into a presentation space using the current pattern attributes.
GpiPolyLine	Draws a series of straight lines starting at the current position and connecting the points specified.
GpiPolySpline	Creates a succession of Bezier splines.
GpiQueryCharStringPos	Processes a string as if it is being drawn under the current character attributes using GpiCharStringPos , and returns the positions in the string at which each character would be drawn.
GpiQueryClipBox	Returns the dimensions of the tightest rectangle that completely encloses the intersection of all the clipping definitions.
GpiQueryPel	Returns the color of a pel at a position specified in world coordinates.
GpiRectVisible	Determines whether any part of a rectangle lies within the clipping region of the device associated with the specified presentation space.
GpiRestorePS	Restores the state of the presentation space to the one that exists when the corresponding GpiSavePS is issued.
GpiSavePS	Saves information about the presentation space on a LIFO stack.
GpiSetClipRegion	Defines the region to be used for clipping when any drawing takes place through the specified presentation space.
GpiSetPel	Sets a pel at a position specified in world coordinates using the current (line) color and mix.
GpiWCBitBit	Copies a rectangle of bit-map image data.
Font Functions	
GpiCreateLogFont	Provides the logical definition of a font.
GpiDeleteSetId	Deletes a logical font or bit-map tag.
GpiLoadFonts	Loads one or more fonts from the specified resource file.
GpiQueryFontMetrics	Returns a record providing details of the font metrics for the logical font that is currently selected.
GpiQueryFonts	Returns a record providing details of the fonts that match the specified <i>pszFacename</i> .
GpiQueryNumberSetIds	Returns the number of local identifiers currently in use, referring to fonts or bit maps.
GpiQueryWidthTable	Returns width table information for the logical font identified by the value of the character-set attribute.
GpiSetCP	Sets the default graphics code page.
GpiUnloadFonts	Unloads any fonts previously loaded from the resource file by GpiLoadFonts .

Table 1-2 (Page 4 of 4). GPI Functions

Presentation Space Functions	
GpiCreatePS	Creates a presentation space.
GpiDestroyPS	Deletes a presentation space.
GpiQueryDevice	Returns the handle of the currently associated device context.
GpiResetPS	Resets the presentation space.
GpiSetPS	Sets the presentation space size, unit, and format.
Region Functions	
GpiCombineRegion	Combines two regions.
GpiCreateRegion	Creates a region for a particular class of device using a series of rectangles.
GpiQueryRegionBox	Returns the dimensions of the smallest rectangle able to bound the region.
GpiSetRegion	Changes a region to be the logical OR of a set of rectangles.
Transform Functions	
GpiConvert	Converts an array of coordinate pairs from one coordinate space to another.
GpiQueryModelTransformMatrix	Returns the current model transform.
GpiSetModelTransformMatrix	Sets the model transform matrix for subsequent primitives.
GpiSetPageViewport	Sets the page viewport within device space.

Chapter 2. Presentation Spaces and Device Contexts

A presentation space is a data structure maintained by the operating system in which information relevant to the graphic output is stored. The information is related to both the subsequent drawings (such as colors or line styles) and the presentation space resources (such as color tables or fonts).

The first task in a graphics application is to define a presentation space, because so many of the GPI functions must operate within them. A presentation space is required for each output device currently in use by your application, including each separate window on the display screen. In almost all cases, the presentation space is set up to be *device-independent*, because the requirements of each possible output device are so different.

Note: In some cases, it is possible to disassociate a presentation space from 1 device and associate it with another, thereby allowing the presentation space to be shared.

To facilitate the device independence of the PM programming interface, all device-specific information is held in a device context. A device context is a data structure that identifies a particular instance of an output device and contains all the device-specific information, such as the logical name of the device and the presentation driver name. Each separate instance of an output device that you intend to use must be described in a device context. For example, if a single application uses more than 1 window, each of those windows must have its own window device context.

The applicable device context, then, must be *associated* with the presentation space in order to send graphics data to that output device. See "Associating Presentation Spaces with Device Contexts" on page 2-10 for details about how to create these associations.

This chapter describes presentation spaces and device contexts. The following topics are related to the information in this chapter:

- Graphics primitives
- Coordinate spaces
- Transformations
- Graphics segments.

About Presentation Spaces

There are 3 types of presentation spaces:

- Standard Micro
- Cached Micro
- Normal.

Micro Presentation Spaces

You need a *micro presentation space* if your application creates a separate presentation space for each output device *instance* and if its output is a simple drawing.

Although run time memory requirements vary according to the graphics function an application uses, a *typical* micro presentation space graphical application uses

315KB of GPI, engine, and display memory resource (code and data). A micro presentation space cannot be reassociated with a different device.

Standard Micro Presentation Space

Use a *standard micro presentation space* for drawing on any type of output device, provided the presentation space is not in *retain mode* or *retain and draw mode*.

There are 2 different functions you can use to access a standard micro presentation space. Table 2-1 lists both functions and their considerations.

Table 2-1. Functions that Create Standard Micro Presentation Spaces		
Function Name	Usage	Closing Function
GpiCreatePS	Accepts an anchor handle, a device-context handle, and a presentation space size as parameters. Creates both normal presentation spaces and micro presentation spaces.	GpiDestroyPS destroys a presentation space and releases all resources owned by the presentation space.
WinGetScreenPS	The presentation space represents the entire display screen. Warning: Exercise caution when using this function as the graphic output can overlap individual windows.	WinReleasePS

Cached Micro Presentation Space

The window manager maintains a cache of micro presentation spaces for windows on a display screen. The cache is provided for applications that use a large number of windows, and where each window requires a temporary presentation space-device context pair for a short sequence of output operations. These presentation spaces belong to the system rather than to your application, and are allocated only on a temporary basis.

Cached micro presentation spaces are provided by the window system rather than by the GPI; their use is synchronized with other window activities. For example, you need not associate a cached micro presentation space with the display screen; the window manager does this for you.

Cached micro presentation spaces offer the best system performance because, unlike normal presentation spaces and standard micro presentation spaces, they are not permanently allocated to an application. The PM programming interface is a model interface, however, cached micro presentation spaces can be cumbersome to use because all the attributes must be initialized continually.

Use a cached micro presentation space to send output only to a window on the display device. There are 3 different functions you can use to access a cached micro presentation space, each with its own considerations. These functions are listed in Table 2-2 on page 2-3.

Table 2-2. Functions that Obtain Cached Micro Presentation Spaces

Function Name	Usage	Closing Function
WinBeginPaint	Accepts a NULL presentation space handle for a cached micro presentation space. The presentation space created by this function is already <i>preassociated</i> with the window device context, making this the easiest function to use. Usually this type of creation is in response to a WM_PAINT message.	WinEndPaint automatically releases the presentation space, no matter what type.
WinGetScreenPS	The presentation space represents the entire display screen. Warning: Exercise caution when using this function as the graphic output can overlap individual windows.	WinReleasePS
WinGetPS	The presentation space can represent the entire desktop, or any other window. The presentation space can be used to process any message, but it must be returned to the cache when message processing is complete.	WinReleasePS

In general, use a cached micro presentation space to process a single paint message when no presentation space information needs to be remembered between messages. The presentation space must be both obtained and released during the processing of that message. All application information stored in a cached micro presentation space is lost as soon as it is released to the cache.

You must provide a window handle on input to WinBeginPaint and WinGetPS. The resulting presentation space is defined specifically for that window, and cannot be reassociated.

The cached micro presentation space always is:

- Defined in *pe/s*—although you can change the units using GpiSetPS
- Formatted GPIF_LONG
- Given a suitable size by the system.

When you finish using a cached micro presentation space, you do not have to disassociate it from the window device context because WinReleasePS or WinEndPaint performs the disassociation. This makes the cached micro presentation space available for use in other windows. The presentation space itself cannot be deleted.

Cached micro presentation spaces are used serially. The next time you need a cached presentation space, access a new 1 using the appropriate function. Each time you get a cached micro presentation space, graphics attributes are reset to their default values.

Normal Presentation Spaces

You must use a *normal presentation space* if you require your application to use the same presentation space to send output to multiple devices (a display screen and printer, for example), or if it uses the segment and retained-drawing functions to generate complex drawings.

There is only 1 function you can use to create a normal presentation space:

Table 2-3. Functions that Create Normal Presentation Spaces		
Function Name	Usage	Closing Function
GpiCreatePS	Accepts an anchor handle, a device-context handle, and a presentation space size as parameters. Creates both normal presentation spaces and micro presentation spaces.	GpiDestroyPS destroys a presentation space and releases

If a normal presentation space is used, or if metafileing is carried out, more run time memory is used than if a micro presentation spaces was used. A normal presentation space requires 114KB more than a micro presentation space.

Modal Graphic Systems

The graphic output sent to the presentation space is created by graphic primitives, such as straight or curved lines. How those primitives appear depends in part on the *mode* of the presentation space.

In modal systems, certain information that modifies a graphic primitive is established before you issue the instruction to draw the primitive. For example, Figure 2-1 is an example of a red, solid line being drawn from (4,3) to (9,7). With PM's modal interface, the only value you need to supply to the GpiLine function is the end point (9,7).

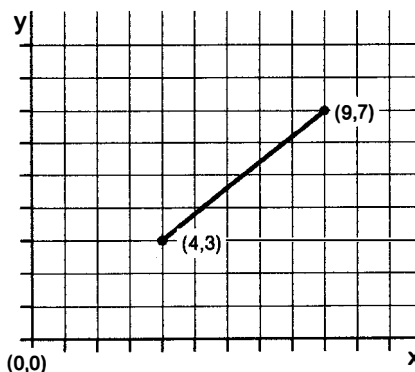


Figure 2-1. Line Drawn with the GpiLine Function.

The line is drawn automatically with the current color, red, in the current line type, solid, and starting from the current position, (4,3). There are separate functions for specifying the current color, line type, and position.

In nonmodal graphics systems, you supply information relevant to an instruction when you issue that instruction. For example, you could issue a line-drawing instruction in which you would specify that the line is to be green, dotted, and drawn from point *a* to point *b*. Those values or *attributes* are bound to a particular drawing instruction, and affect no other. If you are going to continue the line from point *b* to point *c*, you must again specify “green,” “dotted,” “point *b*,” and finally, “point *c*.”

For the PM programming interface, the default values for the graphics primitives (or current values if the defaults have been modified) are stored in the presentation space along with the current position. While at first it might seem like many functions are necessary to perform 1 step, that is, to draw the line, a modal graphics system actually saves resources. After you perform the first line draw, chances are that if you draw a second or third line, the line attributes are not likely to change, that is, the mode of the presentation space remains the same. Thus, the constant repetition of attributes in a nonmodal graphics system actually uses more resources if the desired output requires more than 1 or 2 graphic functions.

It is possible to define graphics, and store their definitions, without sending them to a screen or a printer. In these situations, the concept of *attribute currentness* becomes relevant. The color of a line on a screen, for example, is the color that was current when you defined the line. This is not always the color that is current when the line is drawn.

Current Position

The *current position* is the *world-coordinate space* position at which the next drawing request begins; it is part of the data stored in the presentation space. When you create or obtain a presentation space, the current position is set to the origin (0,0). The current position also is reset to (0,0) whenever a presentation space is associated with a different device context.

Most drawing requests update the current position. If you draw a line from the origin to (3,5), for example, the current position is updated to (3,5). The current position also can be updated explicitly by `GpiSetCurrentPosition` or `GpiMove`. If you do not want the first drawing in a presentation space to start at (0,0), call `GpiSetCurrentPosition` first.

Primitive Attributes

The graphics primitives and the functions that draw them are discussed in Chapter 3 through Chapter 9. The primitive's function allows the attributes of the primitive to be set for all subsequent issuance of that primitive. The attributes are stored in data structures called *bundles*. There are functions that allow you to set the attributes of the primitives before the drawing of any primitive is performed. `GpiSetAttrs` allows you to not only specify the primitive for which the attributes are going to be set, but also, the settings for the specific attributes. `GpiQueryAttrs` returns the current values of specified attributes for specified primitives.

Two other functions, `GpiSetAttrMode` and `GpiQueryAttrMode`, enable you to define or query the attribute mode settings, `AM_PRESERVE` and `AM_NOPRESERVE`. The `AM_PRESERVE` setting preserves the value of an attribute any time that attribute is changed. The previous value can be recovered easily if, for example, you need 6 dotted lines, 1 solid line, then 6 more dotted lines. The `AM_NOPRESERVE` setting discards the previous value of an attribute once the value is changed. The values of the attributes are presentation space resources.

Using Presentation Spaces

You can use presentation-space functions to:

- Create a normal, micro, or cached micro presentation space
- Delete, save, or restore a presentation space
- Define or determine the presentation space attributes.

Obtaining and Creating a Presentation Space

Refer to the *OS/2 2.0 Programming Guide, Volume II* for details and sample code on how to obtain and create presentation spaces.

Deleting a Presentation Space

A presentation space consumes a considerable amount of memory; you should delete it when your application no longer requires it. Delete either a normal or micro presentation space with `GpiDestroyPS`.

When you finish using a cached micro presentation space that you accessed with `WinGetPS`, release it with `WinReleasePS`. You need not delete a presentation space that you accessed with `WinBeginPaint`. PM does this for you when you call `WinEndPaint`.

Saving and Restoring a Presentation Space

You can save certain attributes and resources of a presentation space, modify its fields, draw in the modified presentation space, then restore it with `GpiSavePS` and `GpiRestorePS`. When you call `GpiSavePS`, the graphics engine copies the following items from the current presentation space onto a special stack:

- Primitive attributes
- Transformation matrixes
- Viewing limit
- Clip path
- Clip region
- Current position
- Loaded logical color table
- Loaded logical font.

You can push the contents of a presentation space on the stack as many times as is necessary. `GpiRestorePS` *pops* the contents of a presentation space off the stack.

Presentation Spaces Review

Table 2-4 summarizes the features and restrictions of each type of presentation space.

Table 2-4 (Page 1 of 2). Presentation Space Features and Restrictions			
Feature/Restriction	Normal	Standard Micro	Cached Micro
Device types supported	Any device	Any device	Video-display window only
Number of supported devices	Multiple	One	One
Association	Associate and disassociate as required.	Associate at creation; cannot disassociate.	N/A.

Table 2-4 (Page 2 of 2). Presentation Space Features and Restrictions

Feature/Restriction	Normal	Standard Micro	Cached Micro
Retained graphics	Yes	No	No
Available GPI functions	All functions	All except segment functions	All except segment functions
Memory considerations	Highest memory usage	Medium memory usage	Quickest allocation

About Device Contexts

Device contexts link presentation spaces to devices by converting device-independent presentation space information into device-dependent information. This conversion occurs in the *presentation driver*, a low-level program that is transparent to the application. A presentation driver is a device-driver code that converts the application commands into output commands specific for each output device.

After creating a device context, it must be associated with a presentation space before the drawing (or primitives) can be printed or displayed. When a presentation space and a device context are associated with each other, any output you direct to the presentation space is directed automatically at the device context and displayed on the physical device. For example, before you can draw a picture in a display window, the picture's graphics presentation space must be associated with the window's device context.

Device contexts also give applications access to important device information such as screen dimensions or printer capabilities.

As with presentation spaces, there are 3 types of device contexts:

- Cached
- Window
- Normal.

Cached Device Contexts

A *cached device context* is 1 that already is associated with a cached presentation space when that space is obtained from the cache.

Window Device Contexts

Window device contexts are a special type, in that they are the only device contexts obtained with `WinOpenWindowDC`. As their name implies, the output device they represent is a display window. `WinOpenWindowDC` accepts a window handle as input, and returns the window device context handle. The window device context handle can then be associated with a standard micro presentation space or a normal presentation space.

The window device context is closed automatically when its associated window is deleted.

Normal Device Contexts

A *normal device context* (also called a *standard* or *noncached* device context) links a presentation space with any nonwindow output device. Use DevOpenDC to obtain a device context handle to 1 of 6 normal device contexts listed in Table 2-5. Each of the 6 normal device contexts must be closed by DevCloseDC before your application is closed.

Table 2-5. Normal Device Contexts		
DC Type	Purpose	Usage
Queued	Links a presentation space with a printer or plotter shared by multiple applications sending spooled print jobs to the print queue. Queued device contexts store print jobs by using a program, called <i>print spooler</i> , which keeps track of the order in which the jobs arrive at the printer and in which they are printed.	Applications use queued device contexts to offload printing control from the application.
Direct	Links a presentation space with a printer or plotter, directly bypassing the spooler and print queue. A direct device context is used by the spooler to process jobs as they are removed from the print queue.	Applications normally do not use direct device contexts, unless they are avoiding the queue (for security reasons) or going directly to a dedicated machine.
Information	Links a presentation space with a printer or plotter, directly enabling device information to be queried, but producing no output on the device.	An application can use an information device context with lower memory overhead, rather than use a direct device context, which could provide the same information.
Memory	Links a presentation space with a bit map.	Applications use memory device contexts for drawing to the bit map and using it as a source or target of BitBlt operations.
Metafile	A special device context that enables a picture output to its associated presentation space to be recorded in a metafile for interchange for future use.	Only applications that use metafiles use metafile device contexts.
Metafile_NoQuery	Functionally identical to metafile; however, querying of presentation space attributes is not allowed.	If attributes of the presentation space are not to be queried, this device context offers improved performance over metafile.

Using Device Contexts

In addition to associating or disassociating presentation spaces with device contexts, you can use device context functions to:

- Obtain or create a device context
- Associate a device context with a presentation space
- Close a device context
- Retrieve information about a device's capabilities.

See "Associating a Presentation Space" on page 18-21 for details on associating and disassociating device contexts and presentation spaces.

Creating a Device Context

You can create a normal device context by calling `DevOpenDC`. This function requires that you specify 1 of the 5 normal types. It also requires that you pass certain device-initialization data, including a logical address, the device-driver name, device-driver data, a description of the device type, and information about the queue (if the device is a queued device). The device-initialization data is passed in a `DEVOPENSTRUC` structure.

Figure 2-2 is an example of this structure.

```
typedef struct _DEVOPENSTRUC { /* dop */
    PSZ    pszLogAddress    /* Logical-device address */
    PSZ    pszDriverName    /* Device-driver name */
    PDRIVDATA pdriv        /* Pointer to extra driver data */
    PSZ    pszDataType      /* Type of queued data */
    PSZ    pszComment       /* Optional spooler info */
    PSZ    pszQueueProcName /* Queue-processor name */
    PSZ    pszQueueProcParams /* Queue-processor arguments */
    PSZ    pszSpoolerParams /* Spooler arguments */
    PSZ    pszNetworkParams /* Network arguments */
} DEVOPENSTRUC;
```

Figure 2-2. DEVOPENSTRUC Structure. The last four fields in this structure apply only to queued devices.

Figure 2-3 shows how to create a nondisplay device context for a printer.

```
HDC hdcPrinter; /* Handle of printer device context */
HAB hab; /* Anchor-block handle */
DEVOPENSTRUC dop; /* Device information */

dop.pszLogAddress = "lpt1"; /* Logical-device address */
dop.pszDriverName = "EPSON"; /* Device-driver name */
dop.pdriv = (PDRIVDATA) NULL; /* Pointer to driver data */
dop.pszDataType = "PM_Q_STD"; /* Standard queued data */

hdcPrinter = DevOpenDC(hab,
    OD_DIRECT, /* Direct device type */
    "", /* No data in OS2.INI */
    4, /* Use first 4 fields in dop structure */
    (PDEVOPENDATA) &dop,
    (HDC) NULL);
```

Figure 2-3. Creating a Printer Device Context

Figure 2-4 is an example of how to create a standard device context for a metafile.

```

HDC hdcMeta;                                /* Handles of metafile */
                                           /* and window DCs */
HAB hab;                                    /* Anchor-block handle */
DEVOPENSTRUC dop;                          /* Device information */

dop.pszLogAddress = NULL;                  /* Logical-device address */
dop.pszDriverName = "DISPLAY";            /* Device-driver name */

hdcMeta = DevOpenDC(hab,
                    OD_METAFILE,          /* Metafile DC */
                    "",                   /* No data in OS2.INI */
                    2,                    /* Use first 2 fields in dop */
                    (PDEVOPENDATA) &dop, /* Structure for system info */
                    NULL)                 /* Compatible with screen */

```

Figure 2-4. Creating a Metafile Device Context

Associating Presentation Spaces with Device Contexts

Drawing graphic objects requires a presentation space and a device context to direct output to a specific instance of an output device, such as a display window or a printer. This *association* enables the device context to identify the output device for that presentation space. Further, the device context identifies the particular *instance* of the output device, such as a printer or display window.

A presentation space can be associated with only 1 device context at a time; the reverse is also true: a device context can be associated with only 1 presentation space at a time.

Figure 2-5 on page 2-11, shows how a presentation space is associated with a window device context. It is then disassociated from the window device context and associated with a printer device context. It cannot be associated with both device contexts simultaneously.

```

WM_Create:
    hdcScreen = WinOpenWindowsDC (hwnd);
    phs = GpiCreatePS (...GPIA.Assoc);
    .
    .

WM_COMMAND:

    Case IDM_File PRINT:                                /* Device selection */
        hdcPrinter = DevOpenDC (...);
        GpiAssociate (hps, NULL);                       /* Disconnect from screen */
        GpiAssociate (hps, hdcPrinter);                 /* Connect to printer */
        .
        .                                                /* Output */
        .
        GpiAssociate (hps, NULL);                       /* Disconnect from print */
        GpiAssociate (hps, hdcScreen);                 /* Reconnect to screen */
        .
        .

WM_PAINT:
    WinBeginPaint (hwnd, hps, NULL);
    .
    .                                                /* Output */
    .

```

Figure 2-5. Associating, Disassociating, and Reassociating a Presentation Space

Figure 2-6 shows how to open a window device context and associate it with a normal presentation space.

```

HDC hdcWin;           /* Window device-context handle */
HPS hpsWin;           /* Normal-presentation-space handle */
HWND hwndClient;     /* Client-window handle */
HAB hab;              /* Anchor-block handle */
SIZEL sizlPage;      /* Presentation page */

hdcWin = WinOpenWindowDC(hwndClient);
hpsWin = GpiCreatePS(hab, hdcWin, &sizlPage,
                    PU_LOENGLISH | GPIA_ASSOC);

```

Figure 2-6. Associating a Cached Device Context with a Normal Presentation Space

Note: This type of code is used when the device context is defined before the presentation space.

WinOpenWindowDC can be called only once for a particular window and returns an error if called a second time. WinQueryWindowDC can be used to obtain a window device context previously allocated using WinOpenWindowDC. Figure 2-7 on page 2-12 shows how to create a presentation space with page units of 0.01 inch (PU_LOENGLISH) and associate it with a printer device context. As input to GpiCreatePS, you supply the height and width of the presentation page.

```

HAB hab;          /* Anchor-block handle      */
HPS hpsPrinter;   /* Presentation-space handle */
HDC hdcPrinter;   /* Device-context handle     */
SIZEL sizlPage;   /* Page structure            */
.
.
.
hpsPrinter = GdiCreatePS(hab, hdcPrinter, &sizlPage,
    PU_LOENGLISH | GPIA_ASSOC);

```

Figure 2-7. Creating a Normal Presentation Space

Closing a Device Context

To close a device context that your application opened with `DevOpenDC`, call `DevCloseDC`. However, you should not try to close a device context that you opened with `WinOpenWindowDC`. The operating system does automatically when the associated window is deleted.

Determining Device Capabilities

Once you have created a device context for a particular output device, you can determine the capabilities of that device by calling `DevQueryCaps`. This function retrieves the following information:

- Device technology (whether the device is a *raster* or *vector* device)
- Maximized window dimensions (if the device is a video display)
- Page dimensions (if the device is a printer or plotter)
- Character-box dimensions
- Marker-box dimensions
- Pel resolution
- Color capabilities
- Mix-mode capabilities.

You can use this information, for example, to select fonts, set up the presentation page, or create a new logical color table.

Summary

Table 2-6 summarizes the presentation spaces and device contexts functions.

<i>Table 2-6. Presentation Space and Device Context Functions</i>	
Function Name	Description
DevCloseDC	Closes a device context that was opened by DevOpenDC.
DevOpenDC	Creates a nondisplay device context.
DevQueryCaps	Determines the capabilities of an output device.
DevQueryHardcopyCaps	Determines the form (or paper size) and related information of printers and plotters.
GpiAssociate	Associates the current presentation space with a device context.
	Disassociates the current presentation space by passing a null device context.
GpiCreatePS	Creates the specified type of presentation space.
GpiDestroyPS	Deletes a normal presentation space or standard micro presentation space
GpiMove	Explicitly sets the current position.
GpiQueryPS	Determines the size, units, and other options of a presentation space.
GpiResetPS	Resets the presentation space.
GpiRestorePS	Pops the contents of a saved presentation space off a stack.
GpiSavePS	Saves items from the current presentation space onto an accessible stack.
GpiSetCurrentPosition	Explicitly updates the current position.
GpiSetPS	Sets the presentation page size, units, and format.
WinBeginPaint	Allocates a cached micro presentation space for WM_PAINT processing (if an existing presentation space is not being used) and establishes the update region for the paint.
WinEndPaint	Returns any cached micro presentation space to the cache, and signals completion of the WM_PAINT processing.
WinGetPS	Allocates a cached micro presentation space from the cache.
WinGetScreenPS	Allocates a cached micro presentation space from the cache for the entire display screen.
WinOpenWindowDC	Creates a device context for a display window.
WinQuerySysValue	Copies the size of the client area of the desktop window to the presentation page.
WinQueryWindowDC	Determines the device context for a display window when WinOpenWindowDC has been called previously.
WinReleasePS	Returns a cached micro presentation space to the cache.

Table 2-7 summarizes the data structures used by the presentation space and device context functions.

<i>Table 2-7. Presentation Space and Device Context Structures</i>	
Structure Name	Description
DEVOPENSTRUC	Contains device-initializing data for non-display device contexts.
SIZEL	Specifies the dimensions of the presentation space's presentation page.

Chapter 3. Line and Arc Primitives

Line and arc primitives are graphics building blocks for creating pictures that consist of objects such as polygons, circles, fillets, ellipses, and other geometric figures.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Color and mix attributes
- Area primitives.

About Line and Arc Primitives

Simple drawing applications use line and arc primitives as drawing tools; they are useful, for example, in spreadsheet applications for constructing pie charts, bar charts, and graphs, as shown in the following figure.

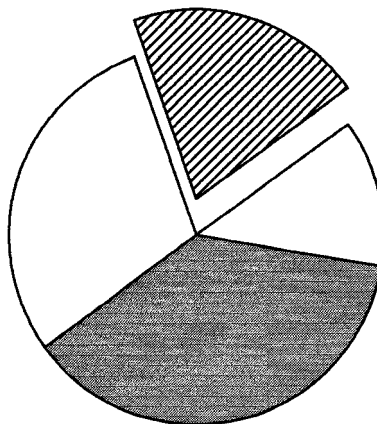


Figure 3-1. Sample Pie Chart Created with Line and Arc Primitives

Following are more illustrations that were drawn using line and arc primitives.

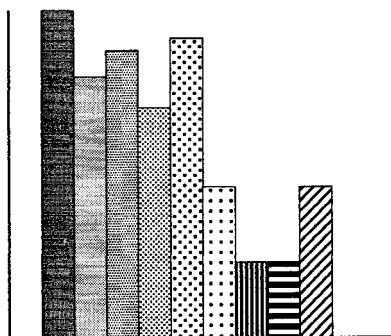


Figure 3-2. Sample Bar Graph Created with Line and Arc Primitives

Computer-aided-design (CAD) applications combine line and arc primitives to draw such complex pictures as schematic diagrams for electrical wiring, blueprints for building sites, and cross-sectional views of machinery.

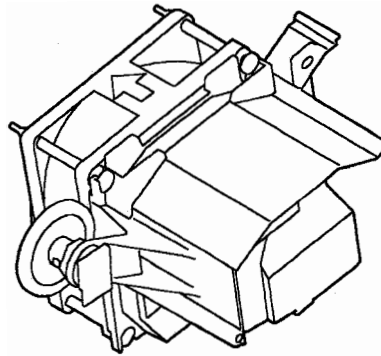


Figure 3-3. Sample Blueprint Created with Line and Arc Primitives

Line and arc primitives are actually two families of primitives that contain many variations of straight lines and curved lines, respectively. They are presented together in this chapter because all the variations are governed by the attributes found in the data structure LINEBUNDLE.

An application draws the line and arc primitives by first issuing GpiMove or GpiSetCurrentPosition, either of which sets the current position to a specified starting point. GpiMove ignores the AM_PRESERVE mode; whereas GpiSetCurrentPosition saves the current position if the AM_PRESERVE mode is set. The more sophisticated function, GpiPolyLineDisjoint, contains its own starting point and does not need to be preceded by GpiMove or GpiSetCurrentPosition.

Prior to calling a line or arc function, the current position can be determined with GpiQueryCurrentPosition.

Attributes of Line and Arc Primitives

The attributes of the line and arc primitives contained in LINEBUNDLE are:

- Line width
- Geometric width
- Line type
- Line end
- Line join
- Line color
- Line mix.

The *line end* and *line join* attributes apply only to lines drawn in a primitive called a *path*. These attributes and paths themselves are explained in Chapter 10, “Paths” on page 10-1.

When an application creates a presentation space, the line and arc attributes are set to the default values shown in Table 3-1.

Table 3-1. Line Attribute Default Values		
Attribute	Default Value	Function that Redefines Attribute
Width	1.0	GpiSetLineWidth
Geometric width	None	GpiSetLineWidthGeom
Type	LINETYPE_SOLID	GpiSetLineType
Color	CLR_NEUTRAL	GpiSetAttrs (LBB_COLOR)
Mix attribute	FM_OVERPAINT	GpiSetAttrs (LBB_MIX_MODE)

Line Width and Geometric Width

The operating system defines two types of lines: cosmetic and geometric, representing two separate concepts.

Cosmetic lines represent the mathematical ideal of a line being an entity with only one dimension—length. When cosmetic lines are drawn, they usually are only one pel wide.

The line width attribute provides a visual method of distinguishing different types of lines. For example, on a map, roads might be drawn thicker than railroad tracks. The line width attribute defines a multiplier to be applied to the normal (or default) line width. Your application can determine the cosmetic line width by calling `GpiQueryLineWidth`.

There are certain devices that do not define the thickness of cosmetic lines in terms of a number of pels. The PM interface interprets the specified width for these devices as shown in the following table:

Table 3-2. Line-Width Attributes as Interpreted by PM	
Identifier	Description
LINEWIDTH_DEFAULT	The default width for the device.
LINEWIDTH_NORMAL	The line width equivalent to a multiplier of 1. Identical to LINEWIDTH_DEFAULT unless changed with <code>GpiSetDefAttrs</code> .
LINEWIDTH_THICK	The line width equivalent to a multiplier greater than 1.

Your application can set the cosmetic line width with either a multiplier or an identifier value using `GpiSetLineWidth`. Since the line represents a widthless ideal, the actual drawn width is a fixed value. The line width remains unchanged when you increase the size of, or zoom in on, a graphics object. The cosmetic line width is not subject to transformations.

In contrast, *geometric lines* represent an area whose thickness is equal to the specified line width. Geometric lines are subject to changes in scale through transformations, in the same manner as geometric figures.

The width of a geometric line does not have a default value. Width is treated as a line attribute for programming convenience, but it actually is a geometric property. A line width is set with `GpiSetLineWidthGeom`; if a width is already specified, that width can be determined with `GpiQueryLineWidthGeom`.

To use geometric line width, the lines are assembled into a path. When you use a path to define wide lines, you can specify the following:

- Geometric width for the lines, using the geometric line width attribute,
- Style of the line ends, using the line end attribute
- Style of the junctions of the lines, using the line join attribute.

These three attributes apply only to geometric lines drawn using paths.

Line Types

Line type, also called *line style*, defines the way a line or arc is drawn: as a solid line, a series of dashes, a series of dots, or a combination of dashes and dots.

The line type for all subsequent line primitives is selected using `GpiSetLineType`. If you want to change the line type to `LINETYPE_DOT`, for example, any line or arc primitive subsequently drawn is drawn as a dotted line. Table 3-3 illustrates the nine standard line types provided by PM.

Note: You cannot define other line types for PM applications.

Table 3-3. Standard Line Types		
Type	Identifier	Long Value
Dotted line	LINETYPE_DOT	1L
Short-dashed line	LINETYPE_SHORTDASH	2L
Dash-dot line	LINETYPE_DASHDOT	3L
Double-dotted line	LINETYPE_DOUBLEDOT	4L
Long-dashed line	LINETYPE_LONGDASH	5L
Dash-double-dot line	LINETYPE_DASHDOUBLEDOT	6L
Solid line	LINETYPE_SOLID	7L
Invisible line	LINETYPE_INVISIBLE	8L
Alternate pels on	LINETYPE_ALTERNATE	9L

The default line type (`LINETYPE_DEFAULT`) is identical to the `LINETYPE_SOLID` type, and has a long value of 0L. The error line type (`LINETYPE_ERROR`) has a long value of -1L. Figure 3-4 illustrates the nine line types. Your application can determine the current line type using `GpiQueryLineType`.

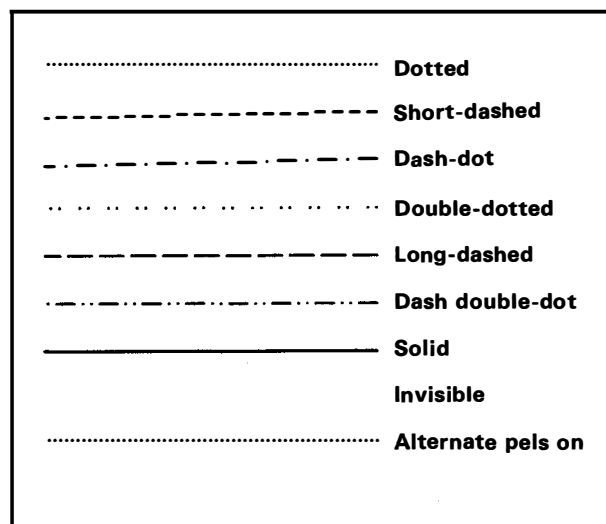


Figure 3-4. The Nine Line Types

Line Color and Mix Attributes

The color attribute defines the color used to draw a primitive or an object. The mix attribute determines how the color of a primitive or an object is combined with the color of the drawing surface, or any other objects on the surface. Both attributes also are described in Chapter 7, "Color and Mix Attributes."

The line color defines the color used to draw the output from any of the operating system's line functions. When a presentation space is created, the line color initial default is black. Unlike certain other primitives, the line and arc primitives do not have a background color. The current color of the drawing surface automatically plays a greater role in the appearance of line and arc primitives than in those primitives that do have a background color. Specifically, when an application draws a dotted or dashed line, the color that appears between the dots or dashes is the current drawing-surface color, as shown in Figure 3-5.

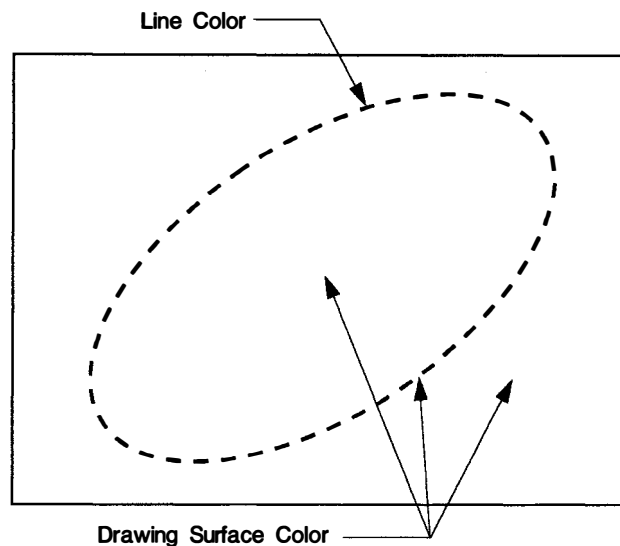


Figure 3-5. Line and Arc Primitives. Line and arc primitives have only a color attribute for the actual line. The mix attribute controls the combination of line color with drawing surface color.

When a presentation space is created, the line mix attribute initial default is `FM_OVERPAINT`. The *overpaint* mix attribute specifies that the line color is not modified by the color of the drawing surface. If the line mix attribute is changed, the line color is mixed with colors that are already on the drawing surface.

To specify a new color or mix attribute issue `GpiSetAttrs`. This function accepts as input the type of primitive, for example `PRIM_LINE`, a list of attributes that are to be changed, a list of attributes that are to be set to their default values, and the values for the attributes that are to be changed. `GpiSetAttrs` is useful for specifying colors and mix attributes just for a specific data structure, for example the `LINEBUNDLE` structure. `GpiSetAttrs` also provides some protection against invalid colors as described in *Presentation Manager Programming Reference*.

To determine the current line color and mix attribute issue `GpiQueryAttrs`. This function accepts as input the primitive type and the attributes in question. It returns as output an array of values for the specifically queried attributes.

To reset the default line color and mix attribute, just as with any other attribute specified in the `LINEBUNDLE` data structure, issue `GpiSetDefAttrs`. This function

accepts as input the type of primitive, for example PRIM_LINE, the attributes to be changed, and the values that will become the new default values. The changing of default values is important when working with segments, described in Chapter 12, "Creating and Drawing Retained Graphics" and Chapter 13, "Editing Retained Graphics and Graphics Segments." Changing the default values during a series of drawing functions is not recommended.

The line color and mix attribute also can be specified with GpiSetColor and GpiSetMix, respectively. However, those two functions have the disadvantage of specifying the color and mix attribute for all primitive xxxBUNDLE data structures that have a component for foreground color and foreground mix attribute. The queries, GpiQueryColor and GpiQueryMix, determine the color and mix attribute as specified by GpiSetColor and GpiSetMix. If the line color or mix attribute is specified individually, the aforementioned queries can return a value inconsistent with the current line color or mix attribute.

Line Primitive Family

Table 3-4 describes the three variants of the basic line primitive and the functions that draw them.

<i>Table 3-4. Functions that Draw Straight Lines</i>		
Variants	Function	Description
Lines	GpiLine	Draws a single line from the current position to a specified point.
Polylines	GpiPolyLine	Draws a series of connected lines, from the current position through successive points.
(series of lines)	GpiPolyLineDisjoint	Draws a series of unconnected lines.
Boxes	GpiBox	Draws a rectangular box with one corner at the current position.

When the operating system draws a line, it includes the pels at the starting and ending points of the line. The algorithm used to draw the rest of the line depends on the device driver. For example, a driver for a raster device might use a modified *Bresenham algorithm* to draw a line; but a driver for a vector device, such as a plotter, simply would connect the starting and ending points of the line. In all cases, the result is a line primitive that looks the same from device to device.

Lines

GpiLine draws a line from the current position to a specified end point, as shown in Figure 3-6. After drawing the line, the current position is at the end point specified by GpiLine.

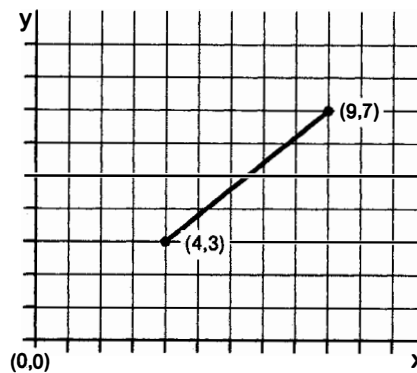


Figure 3-6. GpiLine. The current position is (4,3), and the specified end point of GpiLine is (9,7). When the line has been drawn, (9,7) becomes the current position.

To draw a single point, you can issue GpiLine, with an end point identical to the current position, as shown in Figure 3-7. The current position can be determined using GpiQueryCurrentPosition.

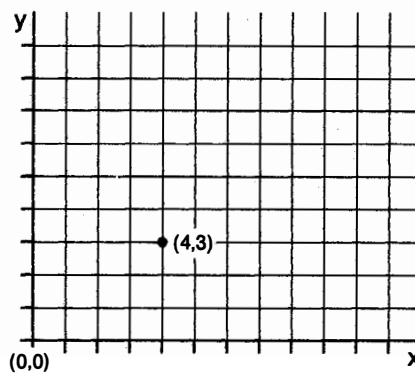


Figure 3-7. GpiLine Used to Draw a Single Point

Polylines

GpiPolyLine draws a sequence of connected lines, starting at the current position and passing through a series of specified coordinate positions as shown in Figure 3-8. After drawing the series of lines, the current position is at the end point of the last line specified by GpiPolyLine.

If you are drawing a graph that has five connected lines, you can issue GpiPolyLine once rather than GpiLine five times. GpiPolyLine accepts as input a number of points and an array of point coordinates.

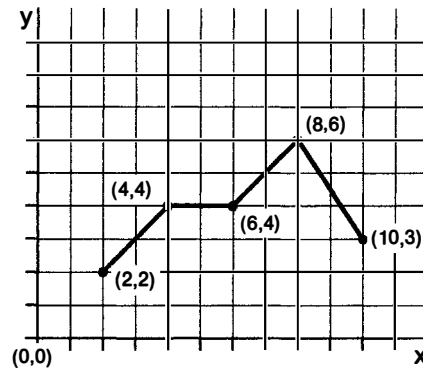


Figure 3-8. GpiPolyLine. The starting position is (2,2), and the polyline is drawn through (4,4), (6,4), (8,6), and (10,3). The new current position is (10,3).

GpiPolyLineDisjoint is a new function that eliminates having to use a series of GpiMove and GpiLine functions to draw multiple lines that are not connected. GpiPolyLineDisjoint accepts as input an even number, the number of points, and an array of point coordinates. The first point in a point-pair is the starting point; the second, the end point of that line segment. Upon completion, the end point of the final line becomes the current position. Figure 3-9 shows the results of a GpiPolyLineDisjoint call.

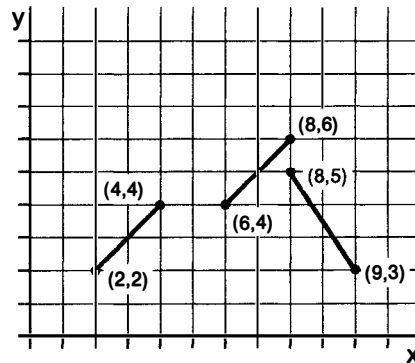


Figure 3-9. GpiPolylineDisjoint. A graph with discontinuities.

Boxes

At its simplest, GpiBox draws a rectangular box with one corner at the current position and the diagonally-opposite corner at a position that you specify. The sides of the box are parallel to the x- and y-axes. Like GpiPolyLine, GpiBox lets you draw a number of connected lines using a single function rather than four separate GpiLine functions. The current position is unchanged by GpiBox.

Note: The start and end position of any closed shape are always the same; therefore, the current position is unchanged after drawing a closed figure.

The GpiBox primitive is shown in Figure 3-10.

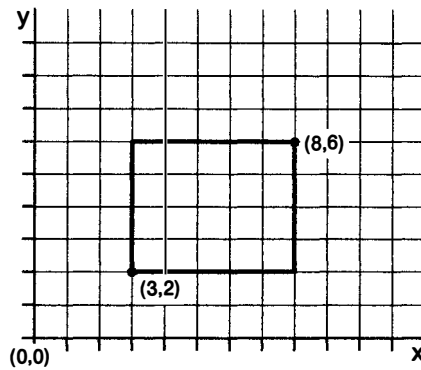


Figure 3-10. The Box. The current position is (3,2) and the corner position is at (8,6). When the box has been drawn, the current position remains at (3,2).

In addition to the corner position, GpiBox accepts as input an option for rounded corners and for filled interior. The PM programming interface rounds the corners of a box by drawing an elliptical section in place of the square corner. Two long values, *IHRound* and *IVRound*, represent the horizontal and vertical length of the full axis of the ellipse used to round each corner. If the two values are equal, a quarter-circle is used for the rounding. If the two values are 0, no rounding is performed. Figure 3-11 shows an example of rounding corners using a quarter-circle.

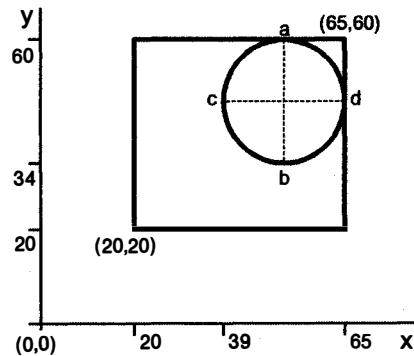


Figure 3-11. Quarter-Circle Box-Corner Rounding. The current position is (20,20). GpiBox is issued with a corner position of (65,60) and an identical vertical and horizontal rounding value of 26.

Figure 3-12 shows the complete result. The intermediate steps are not visible to the user of your application.

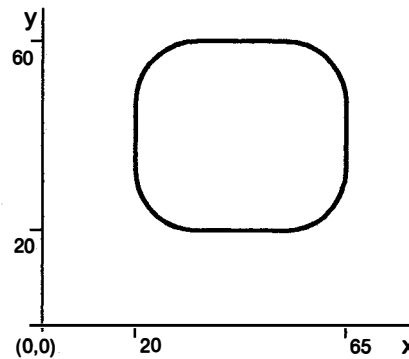


Figure 3-12. Box with Rounded Corners. All four corners are rounded with identical values.

Since GpiBox can be used to define a closed figure, it also accepts as input a long value signifying a filled interior. The long value, *lControl* can be any of the following:

Value	Description
DRO_OUTLINE	Draw the box only.
DRO_FILL	Fill the box interior only.
DRO_OUTLINEFILL	Draw the box and fill its interior.

The pattern that fills the interior and other drawing options are controlled by the data structure AREABUNDLE. Areas are described in Chapter 5, "Area and Polygon Primitives." GpiFullArc is the only other line and arc primitive that can be used to define a closed figure.

Attributes of Arc Primitives

The arc primitive shares width, type, and color and mix attributes with line primitives. For example, if you issue GpiSetLineType to change the line type to LINETYPE_DASHDOT, all subsequent arcs are drawn with a dash-dot line. In addition to the line attributes defined in the LINEBUNDLE data structure, arc primitives in the simple-arc family are influenced by the values in the ARCPARAMS data structure. Arc primitives in the multiple-arc family have a different method of construction and are not influenced by ARCPARAMS.

In terms of geometrical pictures, the simple arcs contain full or partial:

Circles	Closed curves whose center is equidistant from every point on the curve.
Ellipses	Closed curves defined by two fixed points such that the sum of the distances from any point on the curve to the two fixed points is constant.

Multiple arcs contain:

Fillets	Curves that are tangential to the two lines defined by three control points.
Splines	Curves that, given four control points, are tangent to the first and last of three intersecting lines.

There are three simple arc operations that begin with a *unit circle* that lies at the origin of world coordinate space. This unit circle defines the *current arc* on which subsequent full, partial, and 3-point arcs are based. Your application can define the following attributes of the current arc with GpiSetArcParams:

- Shape
- Orientation
- Size
- Drawing direction.

GpiSetArcParams accepts as input a ARCPARAMS data structure that has four parameters (**p**, **q**, **r**, and **s**). Of the four:

- **p** scales in the x-direction
- **q** scales in the y-direction
- **r** and **s** are shear components

The mathematical origins of the parameters are illustrated in Figure 3-13.

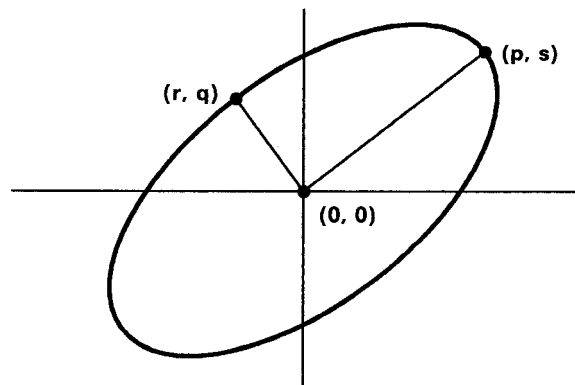


Figure 3-13. The ARCPARAMS Values in World Coordinates. The values were derived from the major and minor axis of an ellipse.

An application can determine the current arc parameters with GpiQueryArcParams, which copies the current arc parameters to their corresponding fields in the supplied ARCPARAMS structure. The application can set the arc parameters with GpiSetArcParams. This function accepts as input a copy of the ARCPARAMS structure that contains the new arc parameters. The default values of **p**, **q**, **r**, and **s**, unless changed by GpiSetDefArcParams, define a unit circle:

- **p** and **q** are 1
- **r** and **s** are 0.

The arc parameters define a transformation that is applied to each point on the perimeter of the unit circle. For any point (x,y) on the perimeter of the unit circle, there exists a new point (x',y'), as determined by the following two linear equations:

$$\begin{aligned} x' &= p \cdot x + r \cdot y \\ y' &= s \cdot x + q \cdot y \end{aligned}$$

These parameters form a 2-by-2 matrix,

$$\begin{bmatrix} p & r \\ s & q \end{bmatrix}$$

that scales and shears simple arcs.

Note: This transformation matrix is not related to the general *transformation functions* that move objects through *coordinate spaces*. It is a special

purpose matrix that transforms the shape and size of the imaginary unit circle. The transformed unit circle, also called the *current arc*, is then used to define the shape and size of the simple arc functions in world coordinates.

After an arc has been described in world coordinates, it can be transformed with the transformation functions, just as any other primitive can.

For detailed information about transformations and their use, see Chapter 17, "Coordinate Spaces and Transformations."

A transformation is orthogonal when:

$$(p \times r) + (s \times q) = 0$$

If orthogonal, the line from the origin (0,0) to the point (p,s) is either:

- The radius of a circle
- Half the major or minor axis of an ellipse;

and, the line from the origin to the point (r,q) is either:

- The radius of a circle
- Half the minor or major axis of an ellipse.

An orthogonal transformation does not guarantee that the shape of an object as defined by an application will be the same as the shape of the object on the output device. For example, if the page units in the application are PU_PELS, and the pels on the device are rectangular (but not square), calling GpiFullArc produces an ellipse—not a circle—even when the arc parameters are set to their default values.

The product of the 2-by-2 matrix multiplication influences the direction of any arc based on the current arc, except a 3-point arc. Table 3-5 illustrates this directional influence.

Table 3-5. Direction of the Arc	
If...	GpiFullArc and GpiPartialArc...
(p x q) is greater than (r x s)	Draw the ellipse counterclockwise.
(p x q) is less than (r x s)	Draw the ellipse clockwise.
(p x q) equals (r x s)	Draw a straight line rather than an ellipse.

Simple-Arc Primitive Family

Table 3-6 describes the three variants of a simple-arc primitive and the functions that draw them. All are defined, to some extent, by the *current arc parameters*. As with line primitives, an application draws simple-arcs by first issuing GpiMove or GpiSetCurrentPosition to set the current position.

Table 3-6. Functions that Draw Simple Arcs		
Variants	Function	Description
Full arcs	GpiFullArc	Draws a circle or an ellipse.
Partial arcs	GpiPartialArc	Draws a straight line followed by a section of a circle or ellipse.
3-point arcs	GpiPointArc	Draws an arc through three points.

Full Arcs

GpiFullArc draws a complete circle or ellipse with its center at the current position. The current position remains unchanged. Whether GpiFullArc draws a circle or an ellipse depends on the current arc parameters. When the current arc is a circle, GpiFullArc draws a circle; when it is an ellipse, GpiFullArc draws an ellipse. Figure 3-16 on page 3-15 shows the full arc.

Defining an Ellipse

To define an ellipse as the current arc, issue GpiSetArcParams on which the values (p,s) and (r,q) are the world coordinates of the end points of the major and minor axes of the ellipse. For example, current arc parameters of (18,0) and (0,10) define an ellipse with a major axis of 36 coordinate units and a minor axis of 20 coordinate units. The resultant ellipse is shown in Figure 3-14.

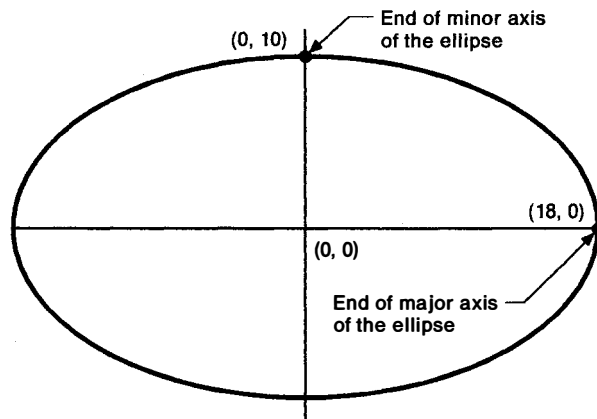


Figure 3-14. The Ellipse

For maximum accuracy, create the axes of an ellipse so that they are at right-angles to each other. You can check this by ensuring that the following equation is always true:

$$p \times r + s \times q = 0$$

So, to check the above example:

$$0 \times 18 + 10 \times 0 = 0$$

You also can define a tilted ellipse as the current arc. None of the current arc parameters for a tilted ellipse will be 0, though you should still ensure that the axes of the ellipse are at right-angles to each other. Figure 3-15 on page 3-14 shows a tilted ellipse defined with current arc parameters of (8,6) and (-3,4).

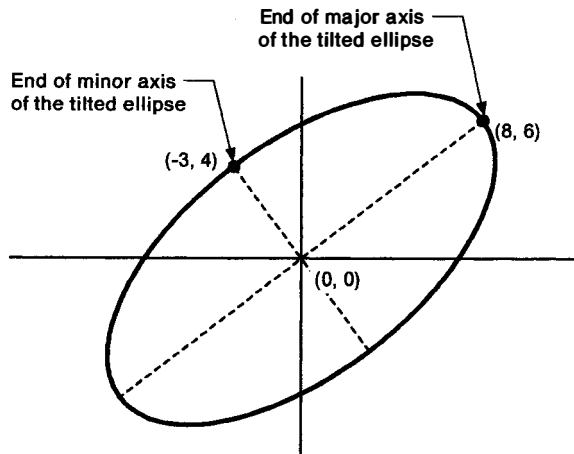


Figure 3-15. Tilted Ellipse

Defining a Circle

To define a circle as the current arc, (r,q) and (p,s) can be any such value that they lie on a circle. For example, if (r,q) and (p,s) are set to $(-4, 3)$ $(3,4)$, the current arc is a circle with a radius of 5 coordinate units. For simplicity, a circle can be defined by specifying current arc parameters where p is the radius of the circle, and $p = q$, $r = 0$, and $s = 0$. For example, to define a circle centered on the origin and with a radius of 10 world-coordinate units, issue `GpiSetArcParams` with the values $(0,10)$ $(10,0)$.

The default values of the current arc parameters are $(0,1)$ $(1,0)$, which define a circle with a radius of one world-coordinate unit (a *unit circle*).

GpiFullArc Input Parameters

`GpiFullArc` accepts as input a multiplier value, so that the size of the full arc is increased or decreased in relation to the current arc. For example, if the current arc parameters define an ellipse whose major axis is 20 coordinate units and whose minor axis is 8 coordinate units, a multiplier of 2 in `GpiFullArc` creates an ellipse whose major axis is 40 coordinate units and whose minor axis is 16 coordinate units.

Because the arc parameters are integers when a fraction is required, for example when rotating an ellipse, greater precision can be obtained by scaling up the required arc parameter values as much as possible, then using a multiplier smaller than 1 to scale the ellipse back down to the required size.

With the default arc parameters defining a circle of 1 world coordinate unit, you can draw a circle of any size by allowing the arc parameters to default, then specifying the radius of the circle with the `GpiFullArc` multiplier. For example, to draw a circle with a radius of 12 coordinate units, issue `GpiFullArc` with a multiplier of 12 and allow the `GpiSetArcParams` to default. Since `GpiFullArc`, like `GpiBox`, can be used to define a closed figure, it also accepts as input a long value signifying a filled interior. The long value, *lControl* can be:

Value	Description
DRO_OUTLINE	Draw the arc only.
DRO_FILL	Fill the arc interior only.
DRO_OUTLINEFILL	Draw the arc and fill its interior.

The pattern that fills the interior and other drawing options are controlled by the data structure AREABUNDLE. See Chapter 5, "Area and Polygon Primitives" for detailed information on using areas.

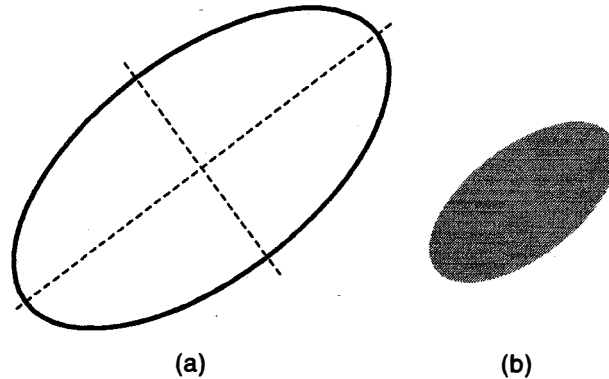


Figure 3-16. The Full Arc. (a) is a tilted ellipse that is defined by the current arc parameters. (b) is drawn using `GpiFullArc` with:

- A multiplier value that reduces the size of the arc relative to the current arc
- An `IControl` value filling the interior with the current area-fill pattern, but not drawing the arc's outline.

Partial Arcs

A partial arc is a section of a full arc defined by the current arc parameters. To draw a partial arc, use `GpiPartialArc`, which draws two separate figures. The first figure is a straight line from the current position to the starting point of a partial arc, and the second figure is the partial arc itself. When the arc has been drawn, the new current position is at the end point of the partial arc.

`GpiPartialArc` accepts as input the center of the current full arc, of which the partial arc is a part, specified in world coordinates. You also can specify a multiplier value to increase or decrease the size of the partial arc in relation to the current full arc.

You also must specify two positive fixed values: a start angle and a sweep angle. If the current full arc is a circle, the start angle is measured counterclockwise from the x-axis of the circle. The intersection of the start angle with the full arc, adjusted by the multiplier value, defines the starting point of the partial arc.

The sweep angle continues the counterclockwise measurement, beginning where the start angle left off. The intersection of the sweep angle with the full arc, adjusted by the multiplier value, defines the partial arc.

If the current arc is not a circle, the start and sweep angle are skewed to the same degree that the ellipse is a skewed circle. Figure 3-17 on page 3-16 shows how the partial arc is constructed.

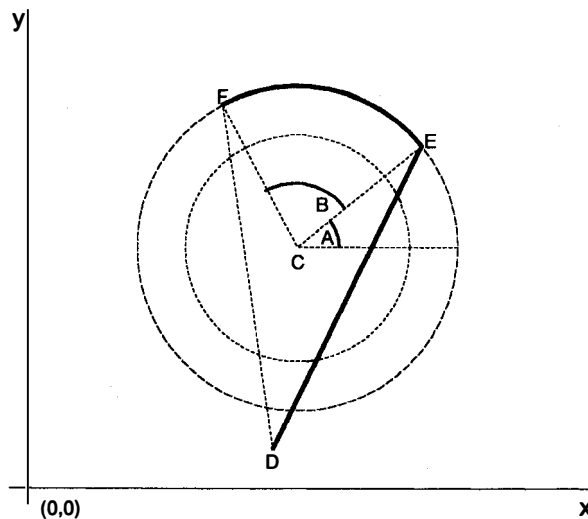


Figure 3-17. The Partial Arc. A is the start angle; B, the sweep angle. C is the center point; D, the current position. EF is the partial arc. F is the new current position.

The inner circle in Figure 3-17 is the arc defined by the current arc parameters. A multiplier has been specified with `GpiPartialArc`, so the partial arc is based on the full arc described by the outer circle. Point C is the center point specified on `GpiPartialArc`. Angle A is the start angle, and angle B is the sweep angle. `GpiPartialArc` therefore, draws a line from the current position (point D) to the start of the partial arc (point E). It also draws EF . Point F is the new current position. The arc is drawn counterclockwise because the current arc parameters define a counterclockwise circle. You can join points F and D with `GpiLine`. This line is drawn automatically if you define the partial arc within a `GpiBeginArea` and `GpiEndArea` bracket.

3-Point Arcs

`GpiPointArc` draws an arc from the current position through an intermediate point to an end point. When the arc is drawn, the current position is at the end point of the arc. You specify both the intermediate point and the end point, and these values determine both the size of the arc and the direction in which it is drawn. The shape and the orientation of a 3-point arc are determined by the current arc parameters. Figure 3-18 shows how a 3-point arc is constructed.

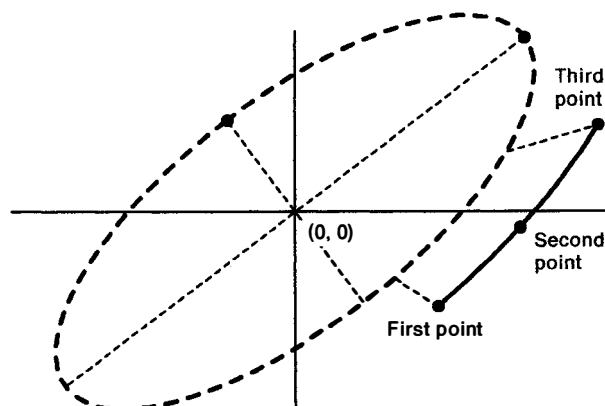


Figure 3-18. The 3-Point Arc.

If the current arc parameters define an ellipse, that ellipse is scaled up or down to fit the 3 points of the arc.

When you want the 3 points of the arc to be points on a circle, allow the current arc parameters to default so they define a unit circle. You do not need to use the values **p** and **q** to specify the radius of the circle, because the radius is determined by the relative positions of the 3 points of the arc.

Multiple-Arc Primitive Family

Table 3-7 describes the 3 variants of the simple-arc primitive and the functions that draw them. As with line primitives, an application draws simple arcs by first issuing **GpiMove** or **GpiSetCurrentPosition** to set the current position. The multiple arcs are the most sophisticated of the arc primitives and their construction does not depend on the current arc parameters.

Table 3-7. Functions that Draw Multiple-Arcs		
Variants	Function	Description
Fillets	GpiPolyFillet	Draws one or more fillets.
	GpiPolyFilletSharp	Draws one or more fillets with varying degrees of sharpness.
Splines	GpiPolySpline	Draws one or more splines.

Fillets

GpiPolyFillet constructs a *fillet* (a curved line) made up of one or more arcs, each of which touches a different straight line. You specify the end points of these straight lines with **GpiPolyFillet**. The lines are not drawn but are used to construct the curve.

The fillet starts at the current position and finishes at the end point of the last line. On the way from the start point to the end point, the fillet is tangential to all intermediate lines at their *midpoints*. When the fillet is drawn, the current position is at the end point of the last construction line. Figure 3-19 is an example of how a fillet is constructed.

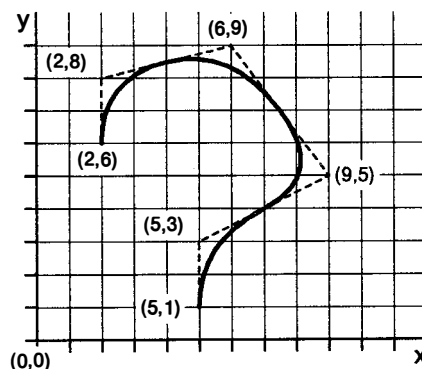


Figure 3-19. The Fillet. The fillet starts at (2,6) and ends at (5,1). The fillet is tangential to the midpoints of the lines from (2,8) to (6,9), from (6,9) to (9,5), and from (9,5) to (5,3).

When you supply only two points, the construction lines of the fillet are drawn from the current position to the first point, and from the first point to the second point. The fillet is drawn from the current position to the second point, and is tangential to the construction lines at those same points.

GpiPolyFilletSharp creates a fillet on a series of connected construction lines. The first fillet in the series is built using two construction lines: one drawn from the current position to point 1 (the control point), and one drawn from point 1 to point 2

(the end point). The fillet is drawn from the current position to the end point, and is tangential to the two construction lines at those points.

GpiPolyFilletSharp also accepts as input a sharpness value. *Sharpness* is a measure of the distance between the fillet and the control point, and is calculated as shown in Figure 3-20.

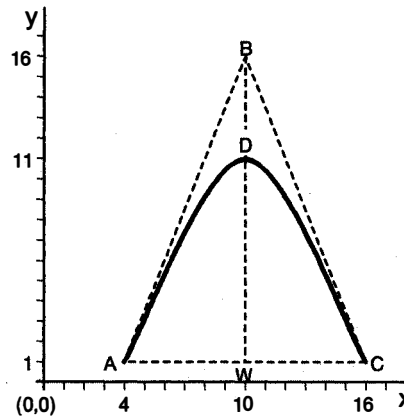


Figure 3-20. Fillet with Sharpness Specified. Point A is the current position, point B is the control point, and point C is the end point. W is the midpoint of the notional line AC. D is the point at which the fillet crosses the notional line WB.

The sharpness of the fillet is the value WD / DB . The line WD is 10 coordinate units, and the line DB is 5 coordinate units; therefore the sharpness value is 2. The sharpness value defines the type of arc, as shown in Table 3-8.

Table 3-8. Fillet Sharpness Values	
A sharpness value of...	Defines...
Greater than 1.0	A hyperbola
Equal to 1.0	A parabola
Less than 1.0	An ellipse

Subsequent fillets start from the end point of the previous fillet, and are constructed using the next two lines in the sequence in exactly the same way. Thus, for each fillet you define one control point, one end point, and one sharpness value. Upon completion, the current position is at the end point of the final construction line in the sequence.

There might be discontinuity of gradient between multiple fillets drawn with GpiPolyFilletSharp. To avoid this, ensure that points B and C of one fillet are on the same construction line as points A and B of the next fillet in the sequence. This concept is illustrated in connection with the spline primitive in Figure 3-22 on page 3-19. Discontinuity of gradient between fillets does not occur when the fillets are drawn with GpiPolyFillet.

Splines

GpiPolySpline creates a succession of one or more *Bezier splines*. The spline also is a curve, but its construction method is different from that of the fillet. As input to this function, you supply three construction points for each spline. The first spline starts from the current position and ends at the third specified point. The two intermediate points are control points for the curve. Subsequent splines start at the end point of the previous spline, have two intermediary control points, and end at the third control point. Figure 3-21 shows the construction method for the spline.

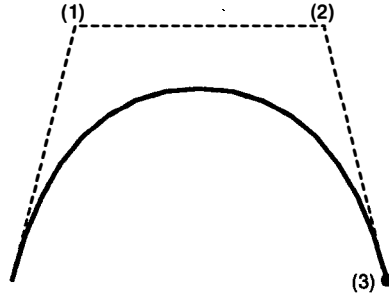


Figure 3-21. *The Spline.* Points (1) and (2) are the control points of the spline, and point (3) is the end point.

To avoid discontinuity of gradient between the end of one spline and the start of the next, ensure that the last two construction points of the first spline and the first two construction points of the second spline are positioned along a single construction line. This concept is shown in Figure 3-22.

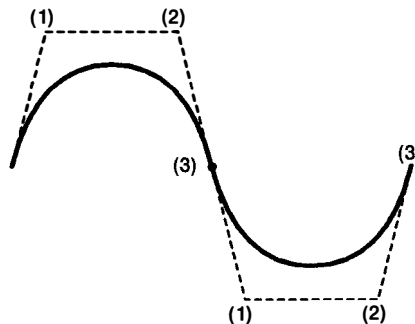


Figure 3-22. *Spline with No Discontinuity of Gradient.* The last two points of the first spline (points 2 and 3) are positioned along the same construction line as the first two points (current position and point 1) of the second spline.

Using Line and Arc Primitives

This section explains how to:

- Draw a straight line
- Create a “rubber-banding” effect with straight lines or arcs
- Draw a circle, ellipse, fillet, or spline.

Drawing a Straight Line

To draw a straight line, set the current position with `GpiMove` or `GpiSetCurrentPosition`. Set the end point of the line by filling in a `POINTL` structure, then draw the line with `GpiLine`. Figure 3-23 shows how to draw a straight line in a PM application.

```
#include <os2.h>

BOOL DrawLine(HPS hps, LONG xStart, LONG yStart, LONG xEnd, LONG yEnd){
    POINTL ptl;                                /* Point structure */

    ptl.x = xStart;                             /* Loads starting x-coordinate */
    ptl.y = yStart;                             /* Loads starting y-coordinate */
    GpiMove(hps, &ptl);                         /* Sets current position */
    ptl.x = xEnd;                               /* Loads ending x-coordinate */
    ptl.y = yEnd;                               /* Loads ending y-coordinate */
    if (GpiLine(hps, &ptl) == GPI_OK){
        return TRUE;                           /* Draw Line. */
    } /* if */
    else return FALSE;
} /* DrawLine */
```

Figure 3-23. Drawing a Straight Line

The second argument of `GpiMove` is the address of a `POINTL` structure that contains coordinates of the line's starting point. The second argument of `GpiLine` is the address of another `POINTL` structure that contains the coordinates of the last point on the line.

Creating a Rubber-Banding Effect with a Straight Line

When lines are drawn with a rubber-banding effect, two things happen: the original line (if one exists) is erased; and a new line is drawn in its place. This process typically takes place each time the mouse is dragged and continues until the mouse button is released. The quickest way to erase the original line (having ensured that it was drawn using mix attribute `FM_XOR`) is to redraw it using mix attribute `FM_XOR`.

Figure 3-24 shows how to create this effect.

```
#define INCL_WININPUT
#define INCL_GPITRANSFORMS
#define INCL_GPIPRIMITIVES
#include <os2.h>
HPS hps; /* Presentation-space handle */
LONG curr_color;

MRESULT EXPENTRY wpGeneric(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    static POINTL ptlStart; /* Starting point of line */
    static POINTL ptlNew; /* Ending point of line */
    static POINTL ptlPrev; /* Previous end point of line */
    static BOOL fDraw; /* Line-drawing flag */

    switch (msg) {
        case WM_BUTTON1DOWN: /* User begins drawing */
            GpiSetColor(hps, CLR_GREEN);
            ptlStart.x = (LONG) (LOUSHORT(mp1));
            ptlStart.y = (LONG) (HIUSHORT(mp1));
            GpiConvert(hps, CVTC_DEVICE, CVTC_WORLD, 1L,
                &ptlStart);
            ptlPrev.x = ptlStart.x;
            ptlPrev.y = ptlStart.y;
            GpiMove(hps, &ptlStart);
            fDraw = TRUE;
            return ((MRESULT) TRUE);

        case WM_MOUSEMOVE: /* User draws line */
            if (fDraw) {
                ptlNew.x = (LONG) (LOUSHORT(mp1));
                ptlNew.y = (LONG) (HIUSHORT(mp1));
                GpiConvert(hps, CVTC_DEVICE, CVTC_WORLD, 1L, &ptlNew);
                curr_color = GpiQueryColor(hps);
                GpiSetMix(hps, FM_XOR);
                if ((ptlStart.x != ptlPrev.x)
                    || (ptlStart.y != ptlPrev.y)) {
                    GpiMove(hps, &ptlStart);
                    GpiLine(hps, &ptlPrev);
                } /* if */
                if ((ptlStart.x != ptlNew.x)
                    || (ptlStart.y != ptlNew.y)) {
                    GpiMove(hps, &ptlStart);
                    GpiLine(hps, &ptlNew);
                    ptlPrev.x = ptlNew.x;
                    ptlPrev.y = ptlNew.y;
                } /* if */
                GpiSetMix(hps, FM_OVERPAINT);
            } /* if */
            return ((MRESULT) TRUE);

        case WM_BUTTON1UP: /* User stops drawing */
            fDraw = FALSE;
            return ((MRESULT) TRUE);
    } /* switch */
} /* wpGeneric */
```

Figure 3-24. Rubber-Banding a Straight Line

Drawing a Circle

When drawing a circle, all the transformations between the world, model, page, and device spaces must maintain square units. This means that your application should select metric, English, or arbitrary page units instead of pels. On most devices, pels are rectangular, but not necessarily square. This also means that the S_x and S_y scaling factors should be equal. If the transformations maintain square units and the arc parameters are set to their default values, `GpiFullArc` produces a circle.

In the example shown in Figure 3-25, if the page units are `PU_LOENGLISH` and the default transformations are set, a circle with a radius of 1/2 inch is drawn.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
HPS hps; /* Presentation-space handle */

MRESULT EXPENTRY wpClient (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    ARCPARAMS arcp; /* Structure for arc parameters */
    POINTL ptlPos; /* Structure for current position */
    FIXED fxMult; /* Multiplier for circle */

    arcp.lP = 1;
    arcp.lQ = 1;
    arcp.lR = 0;
    arcp.lS = 0;
    GpiSetArcParams(hps, &arcp); /* Sets parameters to default */
    ptlPos.x = 100; /* Loads x-coordinate */
    ptlPos.y = 100; /* Loads y-coordinate */
    GpiMove(hps, &ptlPos); /* Sets current position */
    fxMult = MAKEFIXED(50, 0); /* Sets multiplier */
    GpiFullArc(hps, DRO_OUTLINE, fxMult); /* Draws circle */
} /* wpClient */
```

Figure 3-25. Drawing a Circle

The second argument to `GpiFullArc`, `DRO_OUTLINE`, specifies that the operating system should draw only the outline of the circle—rather than filling the interior with the current fill pattern. The third argument, `fxMult`, specifies that the operating system should multiply the size of the circle by 50 units. Because the page units are `PU_LOENGLISH` and the default transformations are set, 50 units is equivalent to 1/2 inch.

Drawing an Ellipse

If you set the world, model, page, and device transformations so that they maintain square units, you can use the arc parameters to transform the shape of the unit circle to an ellipse, and draw this with `GpiFullArc`. The example in Figure 3-26 alters the arc parameter, `p`, by doubling its value, making the ellipse twice as wide horizontally as it is vertically.

In the example shown in Figure 3-26, if the page units are `PU_LOENGLISH` and the default transformations are set, an ellipse with a 2-inch major axis (parallel to the x-axis) and a 1-inch minor axis (parallel to the y-axis) is drawn.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
HPS hps; /* Presentation-space handle */

MRESULT EXPENTRY wpClient (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    POINTL ptlPos; /* Structure for current position */
    FIXED fxMult; /* Multiplier for ellipse */
    ARCPARAMS arcp; /* Structure for arc parameters */

    arcp.lP = 2;
    arcp.lQ = 1;
    arcp.lR = 0;
    arcp.lS = 0;
    GpiSetArcParams(hps, &arcp); /* Sets parameters to default */
    ptlPos.x = 200; /* Loads x-coordinate */
    ptlPos.y = 100; /* Loads y-coordinate */
    GpiMove(hps, &ptlPos); /* Sets current position */
    fxMult = MAKEFIXED(50, 0); /* Sets multiplier */
    GpiFullArc(hps, DRO_OUTLINE, fxMult); /* Draws circle */
} /* wpClient */
```

Figure 3-26. Drawing an Ellipse. Because the arc-parameter fields, `lP` and `lQ`, are set to 2 and 1, the operating system creates an ellipse with a major axis that is twice as long as the minor axis.

Drawing a Pie Slice

The following steps describe how to use `GpiPartialArc` to draw a closed shape bounded by a chord and an arc:

1. Set the current line type to `LINETYPE_INVISIBLE` with `GpiSetLineType`.
2. Issue `GpiPartialArc` with the start angle equal to $\angle B$, and the sweep angle equal to 0. This effectively moves the current position to a point on the current arc, and thereby defines one end of the chord.
3. Select a visible line type with `GpiSetLineType`.
4. Issue `GpiPartialArc`, with the start angle equal to $\angle A$, and the sweep angle equal to $\angle B - \angle A$. Angle B must be greater than angle A. The center point is the same on both `GpiPartialArc` calls.

To fill this partial arc with the current area-fill pattern, you can bracket the `GpiPartialArc` call of step 4 with the `GpiBeginArea` and `GpiEndArea` functions. You should not issue `GpiBeginArea` before step 2.

The effect of this sequence is shown in Figure 3-27.

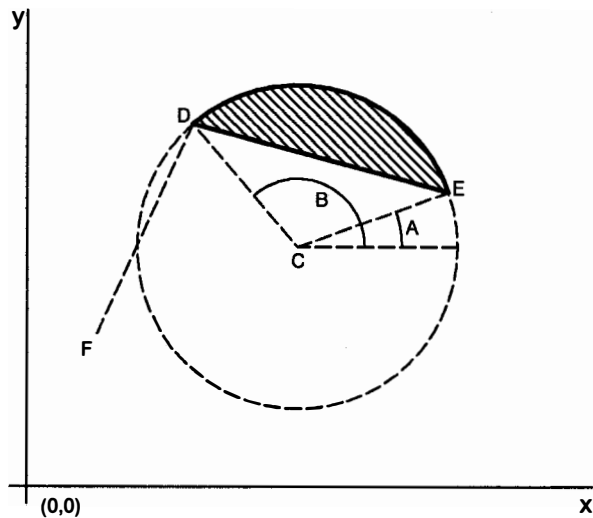


Figure 3-27. Closed Figure Bounded by Chord and Arc

The circle in Figure 3-27 is defined by the current arc parameters. Point F is the current position, and point C is the center of the arc as specified with the first GpiPartialArc call. The first GpiPartialArc call—with the line type set to LINETYPE_INVISIBLE—moves the current position to point D. The second GpiPartialArc call—with the line type set to LINETYPE_SOLID—draws the chord from the current position (point D) to the start point of the arc (point E), and draws $\times$ ED. The partial arc has been defined within an area and has been filled with the current area-fill pattern.

Figure 3-28 shows how to draw a “pie slice” like the one drawn in Figure 3-27.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>

void Figure_516(){
    HPS hps;
    POINTL ptlCenter = {2L, 2L};          /* Coordinates of the center point */
    FIXED fxAngleA = 20L;                  /* Angle A in degrees */
    FIXED fxAngleB = 130L;                 /* Angle B in degrees */

    GpiSetLineType(hps, LINETYPE_INVISIBLE);
    /* Set position to start drawing the arc at angle B. */
    GpiPartialArc(hps, &ptlCenter, MAKEFIXED(1, 0), fxAngleB, 0L);
    GpiSetLineType(hps, LINETYPE_ID);
    GpiBeginArea(hps, BA_BOUNDARY | BA_ALTERNATE); /* Fill the area */
    GpiPartialArc(hps, &ptlCenter, MAKEFIXED(1, 0), fxAngleA, fxAngleB-fxAngleA);
    GpiEndArea(hps); /* Cancel area-fill */
}
```

Figure 3-28. Drawing a Pie Slice

Drawing a Fillet

A fillet is tangential to two lines. The curve of the fillet is always tangential to a line drawn between its start and control points and a line drawn between its end and control points.

Figure 3-29 shows an example of how to draw a single curve, using the current position and two control points.

```
#include <os2.h>
HPS hps; /* Presentation-space handle */

MRESULT EXPENTRY wpClient(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    POINTL aptl[2]; /* Structure for control points */

    aptl[0].x = 50; /* Loads x-coord. of first control point */
    aptl[0].y = 50; /* Loads y-coord. of first control point */
    GpiMove(hps, aptl); /* Sets current position */
    aptl[0].x = 75; /* Loads x-coord. of second control point */
    aptl[0].y = 75; /* Loads y-coord. of second control point */
    aptl[1].x = 100; /* Loads x-coord. of third control point */
    aptl[1].y = 50; /* Loads y-coord. of third control point */
    GpiPolyFillet(hps, 2, aptl); /* Draws fillet */
} /* wpClient */
```

Figure 3-29. Drawing a Fillet

When you draw a sharp fillet, the sharpness value controls the shape of the curve, as previously shown in Table 3-8 on page 3-18.

Figure 3-30 shows an example of using a sharpness value of 3, which creates a hyperbolic curve.

```
#include <os2.h>
HPS hps; /* Presentation-space handle */

MRESULT EXPENTRY wpClient(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    POINTL aptl[2]; /* Structure for control points */
    FIXED fxSharpness; /* Sharpness value */

    aptl[0].x = 50; /* Loads x-coord. of first control point */
    aptl[0].y = 50; /* Loads y-coord. of first control point */
    GpiMove(hps, aptl); /* Sets current position */
    aptl[0].x = 75; /* Loads x-coord. of second control point */
    aptl[0].y = 75; /* Loads y-coord. of second control point */
    aptl[1].x = 100; /* Loads x-coord. of third control point */
    aptl[1].y = 50; /* Loads y-coord. of third control point */
    fxSharpness = MAKEFIXED(3, 0); /* Sets sharpness value */
    GpiPolyFilletSharp(hps, /* Draws fillet */
        2L, aptl, &fxSharpness);
} /* wpClient */
```

Figure 3-30. Drawing a Hyperbolic Curve

Drawing a Spline

When you use `GpiPolySpline` to draw a spline, each curve is tangential to the first and last of three connected lines. Figure 3-31 shows how to draw a spline.

```
#include <os2.h>
HPS hps;                                /* Presentation-space handle */

MRESULT EXPENTRY wpClient(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2){
    POINTL aptl[3];                      /* Structure for control points */

    aptl[0].x = 50;                       /* Loads x-coord. of first control point */
    aptl[0].y = 100;                      /* Loads y-coord. of first control point */
    GpiMove(hps, aptl);                   /* Sets current position */
    aptl[0].x = 75;                       /* Loads x-coord. of second control point */
    aptl[0].y = 200;                      /* Loads y-coord. of second control point */
    aptl[1].x = 100;                      /* Loads x-coord. of third control point */
    aptl[1].y = 0;                        /* Loads y-coord. of third control point */
    aptl[2].x = 125;                      /* Loads x-coord. of fourth control point */
    aptl[2].y = 100;                      /* Loads y-coord. of fourth control point */
    GpiPolySpline(hps, 3L, aptl);         /* Draws spline */
} /* wpClient */
```

Figure 3-31. *Drawing a Spline*

Summary

Table 3-9 summarizes the line and arc primitive functions.

<i>Table 3-9. Line and Arc Primitive Functions</i>	
Function Name	Description
GpiBox	Starting at the current position, draws a rectangular or rounded corner box that, optionally, can be filled.
GpiFullArc	Draws an ellipse or a circle that, optionally, can be filled.
GpiLine	Draws a straight line from the current position.
GpiMove	Sets the current position.
GpiPartialArc	Draws a straight line, followed by a section of a circle or ellipse.
GpiPointArc	Draws an arc through 3 points.
GpiPolyFillet	Draws 1 or more fillets.
GpiPolyFilletSharp	Draws 1 or more fillets with varying degrees of sharpness.
GpiPolyLine	Draws straight lines from the current position to connect a series of vertices.
GpiPolySpline	Draws 1 or more Bezier splines.
GpiQueryArcParams	Determines the current arc parameters.
GpiQueryAttrs	Copies the current line attributes to a LINEBUNDLE structure.
GpiQueryCurrentPosition	Determines the x- and y-coordinates of the current position.
GpiQueryLineType	Determines the current line type.
GpiQueryLineWidth	Determines the current line width.
GpiSetArcParams	Defines the current arc parameters.
GpiSetAttrs	Defines the current line attributes.
GpiSetColor	Defines the current line color.
GpiSetCurrentPosition	Defines the current position and can retain the previous current position.
GpiSetLineType	Defines the current line style.
GpiSetLineWidth	Defines the current line width.
GpiSetMix	Defines the current line mix attribute.

Table 3-10 summarizes the data structures used by the line and arc primitive functions.

<i>Table 3-10. Line and Arc Primitive Structures</i>	
Structure	Description
ARCPARAMS	Contains the current arc parameters.
LINEBUNDLE	Contains the current line parameters.

Chapter 4. Marker Primitives

Marker primitives are graphics objects—such as stars, dots, or crosses—that are used, for example, to indicate the plotted points on a line graph. The following topics are related to the information in this chapter:

- Presentation spaces
- Line and arc primitives
- Color and mix attributes
- Fonts.

About Marker Primitives

Marker primitives always are drawn centered over a point. In a designated presentation space, the `GpiMarker` function draws a single marker primitive of the current marker symbol, with its center at a specified position. This position becomes the new current position when the marker is drawn.

Another marker function, `GpiPolyMarker`, draws multiple marker primitives in the designated presentation space. Each marker primitive is centered over a position specified in an input array to `GpiPolyMarker`. All marker primitives drawn by a single call to `GpiPolyMarker` use the same (current) marker symbol. When a series of marker primitives is drawn, the *current position* is the center point of the last marker in the series. Figure 4-1 shows the use of marker primitives in a line graph.

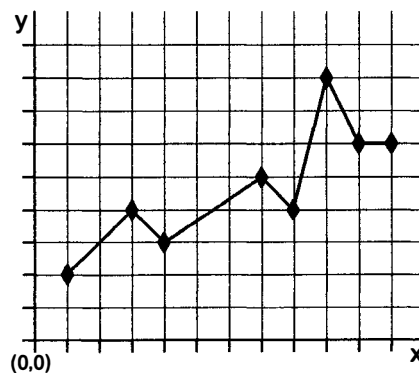


Figure 4-1. Marker Primitives. This example shows a sequence of diamond-shaped marker primitives drawn on a line graph at (1,2), (3,4), (4,3), (7,5), (8,4), (9,8), (10,6), and (11,6). The new current position is at (11,6). The marker portion of this example could have been drawn with a single issuance of `GpiPolyMarker` or with 8 separate `GpiMarker` calls.

Attributes of Marker Primitives

Marker primitive attributes are contained in a data structure called `MARKERBUNDLE`. Following is a list of these attributes:

- Marker symbol
- Marker box
- Marker set
- Foreground color
- Background color
- Foreground mix attribute
- Background mix attribute.

When an application creates a presentation space, the marker attributes are set to the default values shown in Table 4-1.

<i>Table 4-1. Marker Attribute Default Values</i>		
Attribute	Default Value	Function that Redefines Attribute
Marker symbol	Cross	GpiSetMarker
Marker box	Device dependent; equal to the size of one character	GpiSetMarkerBox
Marker set	LCID_DEFAULT	GpiSetMarkerSet Note: If this default is changed, the base marker set cannot be reselected with GpiSetMarkerSet.
Foreground color	Black	GpiSetAttrs (MBB_COLOR)
Background color	Clear	GpiSetAttrs (MBB_BACK_COLOR)
Foreground mix	Overpaint	GpiSetAttrs (MBB_MIX_MODE)
Background mix	Leave alone	GpiSetAttrs (MBB_BACK_MIX_MODE)

Marker Symbols

The current marker symbol is selected from the current marker set using GpiSetMarker. The marker symbol selected for the specified presentation space is used for all subsequent GpiMarker and GpiPolyMarker calls until a new symbol is selected.

Table 4-2 describes the marker symbols provided by the PM programming interface in a base marker set. These symbols are not necessarily available from other marker sets.

<i>Table 4-2. The Base Marker Set</i>		
Symbol	Identifier	Long Value
Cross	MARKSYM_CROSS	1L
Plus sign	MARKSYM_PLUS	2L
Diamond	MARKSYM_DIAMOND	3L
Square	MARKSYM_SQUARE	4L
Six-point star	MARKSYM_SIXPOINTSTAR	5L
Eight-point star	MARKSYM_EIGHTPOINTSTAR	6L
Solid diamond	MARKSYM_SOLIDDIAMOND	7L
Solid square	MARKSYM_SOLIDSQUARE	8L
Dot	MARKSYM_DOT	9L
Small circle	MARKSYM_SMALLCIRCLE	10L
Blank, (Often called the invisible marker)	MARKSYM_BLANK	64L

The default marker symbol (MARKSYM_DEFAULT) is identical to the MARKSYM_CROSS symbol and has a long value of 0L. The error marker symbol (MARKSYM_ERROR) has a long value of -1L.

Figure 4-2 shows the visible marker symbols from the base marker set. Your application can determine the marker set with `GpiQueryMarkerSet`.

×	Cross	+	Plus
□	Square	■	Solid square
◇	Diamond	◆	Solid diamond
✳	Eight-pointed star	★	Six-pointed star
○	Small circle	▪	Dot

Figure 4-2. The Base Marker Set

Marker Box

The *marker box* is a rectangular boundary that defines the horizontal and vertical space occupied by the marker symbol. The marker box is used to center the current marker symbol.

If the current marker set contains *vector* marker primitives (characters outlined by using line and arc functions), changing the size of the marker box changes the size of the marker primitives also. When you change the size of the marker box, a vector marker symbol is scaled up or down automatically to fit the box. Image marker primitives, however, cannot be scaled. If the current marker set contains image characters, the marker box dimensions are fixed, because you cannot alter the size of image markers.

You can use `GpiSetMarkerBox` to change the size of the marker box for a particular presentation space. Set the value of the appropriate `PSIZEF` structure to the required size. The new size is expressed in world coordinates and should not contain any fractional values.

The default size of the marker box can be determined using `DevQueryCaps`. The values, `CAPS_MARKER_WIDTH` and `CAPS_MARKER_HEIGHT`, depend on the currently associated device and the presentation page. Marker box values set with `GpiSetMarkerBox` can be determined using `GpiQueryMarkerBox`.

`GpiSetAttrs` also can be used to change the size of the marker box; `GpiSetDefAttrs` can be used to change the default size of the marker box.

Marker Set

When you create a presentation space, the current marker set, along with other marker attributes, are set to default. The current marker symbol (the cross described earlier) is specified from the supplied marker set. The supplied set of marker primitives contains only image, or raster, marker primitives.

The marker primitives in the default marker set are drawn by setting the color of the pels in the marker box. Within the marker box, the color of the set pels defines the foreground color. The default foreground color is neutral— black on the display screen and on printers.

The color of the pels that are not set defines the background color. The default background color is the background color on the device—white on the display screen, and the loaded paper color on printers. The default background mix is LEAVE_ALONE, or transparent, which means the background color is irrelevant.

If the default set is changed using `GpiSetDefAttrs`, its markers are not accessible. The markers from the default set can be recovered by calling `GpiSetDefAttrs` and specifying the value `LCID_DEFAULT`, (0).

Customizing Marker Sets

If the current marker set does not contain a symbol that suits your application, you can load a new marker set and select a marker symbol from that set. The only way to see the current marker set is to display it on the screen or view a hardcopy of the symbols. You load a new marker set for a specified presentation space by creating a logical font. Then, you select the font as a marker set by specifying the logical font identifier (*lcid*) as an input parameter to `GpiSetMarkerSet`. *lcids* and their values are described in “Selecting the New Current Font.” After selecting the marker set, call `GpiSetMarker` to select a marker from the set. You can set both values simultaneously, with `GpiSetAttrs`.

You can use any font's character set as a marker set and any character within that marker set as a marker. To determine the value that identifies the current marker set, call `GpiQueryMarkerSet`. To determine the value that identifies the current marker character, call `GpiQueryMarker`. You can retrieve both values simultaneously with `GpiQueryAttrs`. You also can create image marker symbols using the Font Editor.

Marker Color and Mix Attributes

The color attribute defines the color used to draw a primitive or an object. The mix attribute determines how the color of a primitive or an object is combined with the color of the drawing surface, or any other objects on the surface. Both attributes also are described in Chapter 7, “Color and Mix Attributes.”

The marker color defines the color used to draw the output from any of the OS/2 operating system marker functions. When a presentation space is created, the marker color initial default is black. Markers are one of the primitives that have a foreground and background color, as shown in Figure 4-3.

For image markers, the colors are determined by the setting of pels. For vector markers, the foreground consists of the arcs and lines that define the marker. The background color appears between the foreground lines. The marker can be solid, or filled, in which case the background color does not appear between the foreground lines.

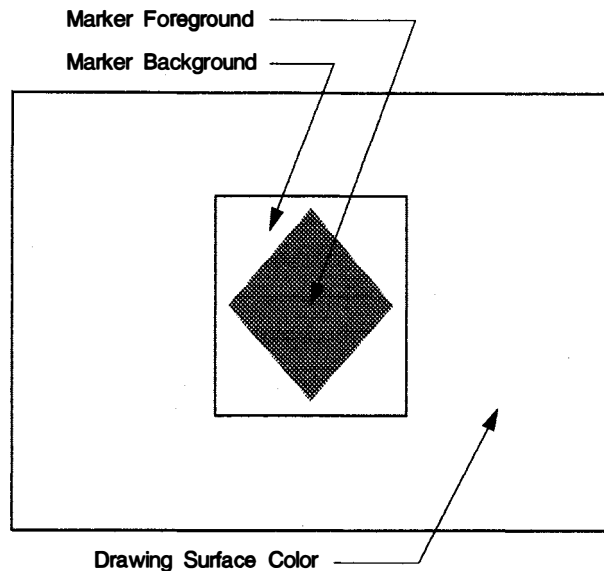


Figure 4-3. Marker Primitives. Marker primitives have both a color and background color attribute. The mix attribute controls the combination of marker color with drawing-surface color, while the background mix attribute controls the combination of the marker box color with the drawing-surface color.

When a presentation space is created, the marker mix attribute initial default is `FM_OVERPAINT`. The *overpaint* mix attribute specifies that the marker color is not to be modified by the color of the drawing surface. If the marker mix attribute is changed, the marker color is mixed with colors that are already on the drawing surface.

The marker background color initial default is `CLR_BACKGROUND`. Usually this is defined by the application to the same color as the drawing surface. The marker background mix attribute initial default is `BM_LEAVEALONE`. The *leave-alone* mix background attribute specifies that the marker background color is not drawn. The box that effectively surrounds the marker appears only if the marker background mix attribute is changed.

To specify a new color or mix attribute call `GpiSetAttrs`. This function accepts as input the type of primitive, for example `PRIM_MARKER`, a list of attributes that are to be changed, a list of attributes that are to be set to their default values, and the values for the attributes that are to be changed. `GpiSetAttrs` is useful to specify colors and mix attributes just for a specific data structure, for example the `MARKERBUNDLE`. `GpiSetAttrs` also provides some protection against invalid colors as described in *Presentation Manager Programming Reference*.

To determine the current marker color and mix attribute call `GpiQueryAttrs`. This function accepts as input the primitive type and the attributes in question. It returns as output an array of values for the specifically queried attributes.

To reset the default marker color and mix attribute, just as with any other attribute specified in the `MARKERBUNDLE` data structure, call `GpiSetDefAttrs`. This function accepts as input the type of primitive, for example `PRIM_MARKER`, the attributes to be changed, and the values that will become the new default values. The changing of default values is important when working with segments, described in Chapter 12, "Creating and Drawing Retained Graphics" and Chapter 13, "Editing Retained Graphics and Graphics Segments." Changing the default values during a series of drawing functions is not recommended.

The marker color and mix attribute also can be specified with:

- GpiSetColor
- GpiSetMix
- GpiSetBackColor
- GpiSetBackMix.

However, these functions have the disadvantage of specifying the foreground and background color or mix attribute for all primitive xxxBUNDLE data structures that have the respective component.

There are four queries that determine the color and mix attribute as specified by GpiSetxxx functions:

- GpiQueryColor
- GpiQueryMix
- GpiQueryBackColor
- GpiQueryBackMix.

If the marker color, marker background color, mix attribute, or background mix attribute were specified individually, the aforementioned queries can return a value inconsistent with the current marker attribute.

Using Marker Primitives

You can use marker functions to:

- Draw a single marker or a series of markers
- Set or determine (query) any combination of the marker bundle attributes including:
 - Determining the *lcid* for the current marker set
 - Selecting a character set as the new marker set
 - Determining the value that identifies the current marker
 - Selecting a character as the new marker
 - Setting or changing the size of the marker box
 - Setting or changing the color of a marker.

Drawing Marker Primitives

You can draw either a single marker or a series of markers using the current marker symbol. To draw a single marker, set the fields in a POINTL structure to correspond to the desired position in world coordinates. Then call GpiMarker, passing it the address of the POINTL structure as the second argument.

To draw a series of markers, set the fields in an array of POINTL structures to correspond to the desired positions in world coordinates. Then call GpiPolyMarker, passing it the number of points in the array as the second argument and the name of the array as the third argument.

Figure 4-4 shows how to draw a graph with GpiPolyLine and GpiPolyMarker.

```
#include <os2.h>
void fncMARK01(void){
    HPS hps;                                /* Presentation-space handle */
    POINTL aptl[6];                          /* Array of points */

    aptl[0].x = 10; aptl[0].y = 15;          /* Assigns points */
    aptl[1].x = 150; aptl[1].y = 30;
    aptl[2].x = 200; aptl[2].y = 32;
    aptl[3].x = 250; aptl[3].y = 70;
    aptl[4].x = 360; aptl[4].y = 120;
    aptl[5].x = 380; aptl[5].y = 98;
    GpiPolyMarker(hps, sizeof(aptl) / sizeof(POINTL), aptl);
                                                    /* Plots points */
    GpiMove(hps, aptl);                        /* Sets current position */
    GpiPolyLine(hps, sizeof(aptl) / sizeof(POINTL), aptl);
                                                    /* Draws lines */
} /* fncMARK01 */
```

Figure 4-4. Drawing a Graph

Selecting a New Marker

Figure 4-5 shows how to check whether the default marker primitive from the default marker set is being used currently; and if so, how to replace the cross with the six-pointed star.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
void fncMARK02(void){
    HPS hps;

    if ((GpiQueryMarker(hps) == MARKSYM_DEFAULT) &&
        (GpiQueryMarkerSet(hps) == LCID_DEFAULT))
        GpiSetMarker(hps, MARKSYM_SIXPOINTSTAR);
}
```

Figure 4-5. Selecting a New Marker

Selecting a New Marker Set

Figure 4-6 is an example of how to load a Helvetica™ vector font, select it as the new marker set, and select the uppercase A as the new marker primitive.

```
#define INCL_GPILCIDS
#define INCL_GPIPRIMITIVES
#include <os2.h>
void fncMARK03(void){
    LONG cHelvFonts, cFonts, lcid, i, j;
    HPS hps;
    FATTRS fattrs;
    FONTMETRICS afm[80];
    MARKERBUNDLE mbnd;

    cHelvFonts = GpiQueryFonts(hps, QF_PUBLIC, "Helv",
    &cFonts, sizeof(FONTMETRICS), (PFONTMETRICS) NULL);
    /* Queries the number of Helvetica fonts. */

    GpiQueryFonts(hps, QF_PUBLIC, "Helv", &cHelvFonts,
    sizeof(FONTMETRICS), afm);
    /* Loads the array of FONTMETRICS structures. */

    for (i = 0; !(afm[i].fsDefn & FM_DEFN_OUTLINE)
    && i < cHelvFonts; i++);
    /* Finds outline font */

    fattrs.usRecordLength = sizeof(FATTRS);
    fattrs.fsSelection = 0;
    fattrs.lMatch = afm[i].lMatch; /* Uses Helvetica outline font */
    for (j = 0; j <= sizeof(afm[i].szFacename); j++)
    fattrs.szFacename[j] = afm[i].szFacename[j];
    fattrs.idRegistry = 0;
    fattrs.usCodePage = 850; /* Uses international code page */
    fattrs.lMaxBaselineExt = 0;
    fattrs.lAveCharWidth = 0;
    fattrs.fsType = 0;
    fattrs.fsFontUse = FATTR_FONTUSE_TRANSFORMABLE;

    GpiCreateLogFont(hps, (PSTR8) NULL, lcid, &fattrs);
    mbnd.usSet = lcid; /* Uses font as marker set */
    mbnd.usSymbol = 'A'; /* Uses capital A as primitive */
    GpiSetAttrs(hps, PRIM_MARKER, MBB_SYMBOL | MBB_SET, 0, &mbnd);
} /* fncMARK03 */
```

Figure 4-6. Selecting a New Marker Set

Changing the Marker Color

Figure 4-7 shows how to set the marker foreground color to green.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
void fncMARK04(void){
    HPS hps;
    MARKERBUNDLE mbnd;

    mbnd.lColor = CLR_GREEN;
    GpiSetAttrs(hps, PRIM_MARKER, MBB_COLOR, 0L, &mbnd);
} /* fncMARK04 */
```

Figure 4-7. *Changing the Marker Color*

Summary

Table 4-3 summarizes the marker primitive functions.

<i>Table 4-3. Marker Functions</i>	
Function Name	Description
GpiMarker	Draws the current marker primitive at a specified position.
GpiPolyMarker	Draws a series of the current marker primitive at a specified position.
GpiQueryAttrs	Determines the current marker bundle attributes.
GpiQueryMarker	Determines the current marker symbol attribute.
GpiQueryMarkerBox	Determines the size of the current marker box.
GpiQueryMarkerSet	Determines the value of the current marker set attribute.
GpiSetAttrs	Sets the marker bundle attributes.
GpiSetMarker	Sets the attribute of the marker symbol.
GpiSetMarkerBox	Sets the dimensions of the marker box. (This function influences vector markers only. The width and height of raster marker primitives are fixed.)
GpiSetMarkerSet	Sets the attribute of the marker set.

Table 4-4 summarizes the data structures used by the marker primitive functions.

<i>Table 4-4. Marker Structures</i>	
Structure Name	Description
POINTL	A structure specifying the values of a single pair of x- and y- coordinates.
PSIZEF	A structure specifying the size of the marker box.
MARKERBUNDLE	A structure of marker primitive attributes.

Chapter 5. Area and Polygon Primitives

An *area* is one or more closed figures that can be drawn filled, outlined, or filled and outlined. If an area includes more than one figure, those figures can be separate or intersecting.

A *polygon*, too, is one or more closed figures that can be drawn filled, outlined, or filled and outlined. Polygons, unlike areas, are limited to figures with straight edges. Polygons also can be separate or intersecting.

The following topics are related to the information in this chapter:

- Presentation spaces
- Line and arc primitives
- Color and mix attributes
- Fonts
- Bit maps
- Paths.

About Area Primitives

Applications can create, outline, and fill areas and can create custom-fill patterns from bit maps or font symbols. Some graphics functions are not valid within an area definition. For example, you cannot include marker, image, or character-string primitives in an area definition. Figure 5-1 is an example of an area.

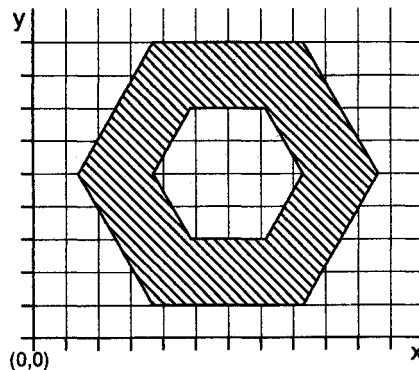


Figure 5-1. An Area. This area comprises two hexagons, one completely enclosed by the other. The area is filled with the current area-fill pattern.

Attributes of Area Primitives

The attributes of the area primitive are contained in a data structure called AREABUNDLE. These attributes are:

- Pattern symbol
- Pattern reference point
- Pattern set
- Foreground color
- Background color
- Foreground mix attribute
- Background mix mode.

When an application creates a presentation space, the area attributes are set to the default values shown in Table 5-1.

Table 5-1. Area Attribute Default Values		
Attribute	Default Value	Function that Redefines Attribute
Pattern symbol	solid	GpiSetPattern
Pattern reference point	(0,0)	GpiSetPatternRefPoint
Pattern set	LCID_DEFAULT	GpiSetPatternSet
Foreground color	Black	GpiSetAttrs (ABB_COLOR)
Background color	Clear	GpiSetAttrs (ABB_BACK_COLOR)
Foreground mix	Overpaint	GpiSetAttrs (ABB_MIX_MODE)
Background mix	Leave alone	GpiSetAttrs (ABB_BACK_MIX_MODE)
Note: If the default (LCID_DEFAULT) for pattern set is changed, the base pattern set cannot be reselected with GpiSetPatternSet.		

Pattern Symbol Attribute

The *current pattern symbol*, also called the *symbol point code*, is selected from the current pattern set with GpiSetPattern. The pattern symbol selected for the specified presentation space is used as the subsequent fill pattern until a new symbol is selected. Applications must not call GpiSetPattern while in an area or a path. If the current pattern set specifies a bit map, this attribute is ignored.

Table 5-2 describes the pattern symbols provided by the PM programming interface in a base pattern set. These symbols are not necessarily available from other pattern sets.

Table 5-2. The Base Pattern Set		
Symbol	Identifier	Long Value
Solid shading decreasing in dots per inch	PATSYM_DENSE1 through PATSYM_DENSE8	1L — 8L
Vertical lines	PATSYM_VERT	9L
Horizontal lines	PATSYM_HORIZ	10L
Lines bottom left to top right	PATSYM_DIAG1	11L
Lines bottom left to middle right	PATSYM_DIAG2	12L
Lines top left to bottom right	PATSYM_DIAG3	13L
Lines top left to middle right	PATSYM_DIAG4	14L
No shading	PATSYM_NOSHADOW	15L
Solid shading	PATSYM_SOLID	16L
Alternate pels	PATSYM_HALFTONE	17L
Cartesian grid	PATSYM_HATCH	18L
Diagonal crosshatch	PATSYM_DIAGHATCH	19L
Blank (often called the clear pattern)	PATSYM_BLANK	64L

The default pattern symbol (PATSYM_DEFAULT) is identical to the PATSYM_SOLID symbol, and has a long value of 0L. The error pattern symbol (PATSYN_ERROR) has a long value of -1L.

Figure 5-2 shows the patterns from the base pattern symbol set. Applications can determine the the current pattern set by issuing GpiQueryPattern.

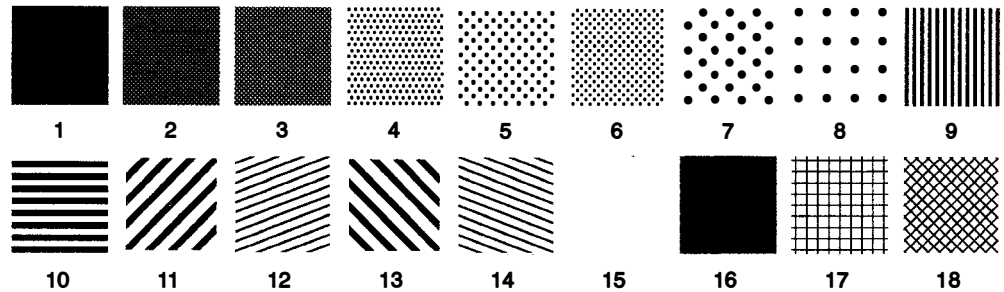


Figure 5-2. The Base Pattern Symbol set.

Pattern Reference Point Attribute

The *pattern reference point* is the point from which the area's fill-pattern spreads horizontally and vertically. The lower-left corner of the pattern is aligned on this point. The pattern reference point has a default value of (0,0), and is expressed in world coordinates. Applications can determine the pattern reference point with GpiQueryPatternRefPoint. Applications can specify the pattern reference point with GpiSetPatternRefPoint; however, this should not be done inside an area. The pattern reference point does not have to be within the boundary of the area. Figure 5-3 shows an area that was filled using the default pattern reference point.

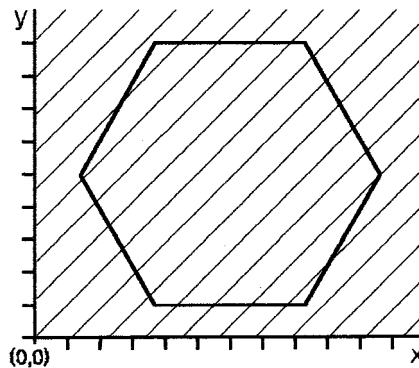


Figure 5-3. The Pattern Reference Point. The pattern of diagonal lines is aligned with the pattern reference point (0,0). When the picture is displayed or printed, only the hexagon will be filled.

Use of the default reference point causes the diagonal lines of the pattern to intersect the area boundary at specific points. If you change the reference point to (0,4), for example, the pattern shifts upward, and the points of intersection with the area boundary are different. Although the reference point is outside the area, when an application displays or prints the picture, only the area is filled.

Pattern Set Attribute

When you create a presentation space, the current pattern set and other area attributes are set to the default. The current pattern—a black solid, as described earlier—is specified from the supplied pattern set. (The supplied set of pattern symbols contains image or raster patterns only.) An area primitive is filled by repeating this pattern vertically and horizontally within the area boundary.

Default Pattern Set

The standard pattern set contains patterns of solid shading with decreasing intensity, and patterns of vertical, horizontal, and diagonal lines. If the default pattern set is changed with `GpiSetDefAttrs`, its patterns are not accessible. The patterns from the default set can be recovered by specifying `GpiSetDefAttrs` with the value `LCID_DEFAULT, (0)`.

Customizing Pattern Sets

An application can create custom patterns by using bit maps or characters and symbols from an image font. When designing custom patterns, consider that the operating system uses only the first 8 bits in the first 8 rows, starting with the lower-left corner of the bit map. The cell size of an image font used as a fill pattern might be too large to include the complete character or symbol.

An application can change the appearance of a fill pattern by changing its alignment in an area primitive with `GpiSetPatternRefPoint`. When the default reference point is set, the operating system aligns the lower-left corner of the fill pattern with the point (0,0) in the application's world space and begins filling the area. If an application adjusts the pattern reference point to (3,2), the operating system aligns the lower-left corner of the fill pattern with the point (3,2) in the application's world space and begins filling the area. By moving the reference point from (0,0) to (3,2), the fill pattern appears shifted up 2 pels and to the right 3 pels.

Custom Fill Patterns from a Bit Map: To create a custom pattern from a hard-coded bit map, an application must use the following steps:

1. Create or load a bit map to obtain a bit-map handle, for example, use `GpiLoadBitmap`.
2. Call `GpiSetBitmapId (ID)` to tag the bit map with a local identifier (`lcid`) from 1 to 254.
3. Call `GpiSetPatternSet (ID)` or `GpiSetAttrs (ABB_SET)` to set the current fill pattern.

The application can now draw the area.

Custom Fill Patterns from a Font Symbol: To create a custom pattern from a character or a symbol in a font:

1. Create a logical image font and assign it an ID, for example use `GpiCreateLogFont`.
2. Call `GpiSetPatternSet (ID)` or `GpiSetAttrs (ABB_SET)`, passing it the local identifier for the font.
3. Call `GpiSetPattern`, passing the value of the code point for a character or symbol in the font.

The application can now draw the area.

Area Colors and Mix Attributes

The color attribute defines the color used to draw a primitive or an object. The mix attribute determines how the color of a primitive or an object is combined with the color of the drawing surface, or any other objects on the surface. Both attributes also are described in Chapter 7, "Color and Mix Attributes."

The area color defines the color used to fill the output from any of the OS/2 operating system area functions, when necessary. The area primitive is the only primitive in which the color can be changed during the drawing of the area; therefore, this attribute, more than the other area attributes, depends on the *current value* as described in "Attribute Currentness" on page 5-12.

When a presentation space is created, the area color initial default is black. The pattern symbol initial default, explained previously, is solid. Areas are one of the primitives that have a foreground and background color, as shown in Figure 5-4; however, the appearance of the area also depends on the pattern symbol.

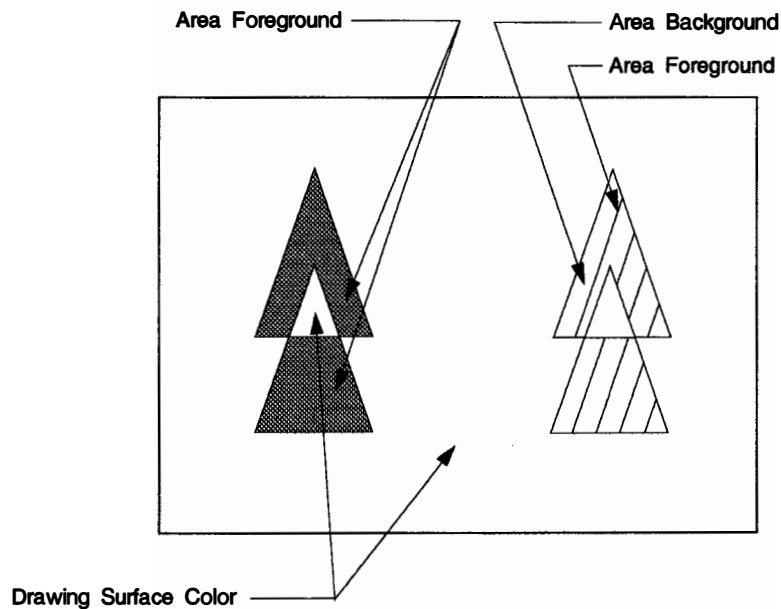


Figure 5-4. Area Primitives. Area primitives have both a color and background color attribute. The background color does not appear if the pattern symbol is solid, and the background color is undefined if the pattern symbol is a customized, multi-colored bit map.

When a presentation space is created, the area mix attribute initial default is FM_OVERPAINT. The *overpaint* mix attribute specifies that the area color is not to be modified by the color of the drawing surface. If the area mix attribute is changed, the area color is mixed with colors that are already on the drawing surface.

When the pattern symbol is solid, the color that fills the area, if necessary, is the current area foreground color. If the pattern symbol is changed to be a pattern of vertical lines, for example, the color of those lines is the current area foreground color. Inside of the area bracket, there can be other colors if these colors are the current color for primitives such as lines or arcs, that are drawn within the area bracket. These colors are not specifically defined or influenced by the area foreground color.

The area background color initial default is CLR_BACKGROUND. Usually this is defined by the application to the same color as the drawing surface. If the pattern

symbol is solid, the area background color *does not* appear. If the pattern symbol is changed to be a pattern of vertical lines, for example, the color in between those lines is the current area background color.

The area background mix attribute initial default is `BM_LEAVEALONE`. The *leave-alone* background mix attribute specifies that the area background color is not drawn. This means that for a nonsolid pattern symbol, the drawing-surface color or the color of an object, not created by the area bracket, but on the drawing surface, shows through the nonsolid pattern. The area background color, for nonsolid pattern symbols, appears only if the background mix attribute is changed to overpaint, `BM_OVERPAINT`.

If you have customized the pattern symbol with a bit map, the color definitions of the area primitive change. The foreground color corresponds to the color of the pels that are specified ON in the fill-pattern bit map. The background color corresponds to the color of the pels that are specified OFF in the fill-pattern bit map. The foreground and background mix attribute do not change.

Note: Background color and mix attribute only apply to monochrome (2-color) fill patterns. The bit set to 0 is defined as the background and the bit set to 1 is defined as the foreground.

A multi-colored bit map is unaffected by the background color and mix attributes.

To specify a new color or mix attribute call `GpiSetAttrs`. This function accepts as input the type of primitive, for example `PRIM_AREA`, a list of attributes that are to be changed, a list of attributes that are to be set to their default values, and the values for the attributes that are to be changed. `GpiSetAttrs` is useful to specify colors and mix attributes just for a specific data structure, for example the `AREABUNDLE`. `GpiSetAttrs` also provides some protection against invalid colors as described in *Presentation Manager Programming Reference*.

To determine the current area color and mix attribute, call `GpiQueryAttrs`. This function accepts as input the primitive type and the attributes in question. It returns as output an array of values for the specifically queried attributes.

To reset the default area color and mix attribute, just as with any other attribute specified in the `AREABUNDLE` data structure, call `GpiSetDefAttrs`. This function accepts as input the type of primitive, for example `PRIM_AREA`, the attributes to be changed, and the values that will become the new default values. The changing of default values is important when working with segments, described in Chapter 12, "Creating and Drawing Retained Graphics" on page 12-1 and Chapter 13, "Editing Retained Graphics and Graphics Segments." Changing the default values during a series of drawing functions is not recommended.

The area color and mix attribute also can be specified with:

- `GpiSetColor`
- `GpiSetMix`
- `GpiSetBackColor`
- `GpiSetBackMix`.

However, these functions have the disadvantage of specifying the foreground and background color or mix attribute for all primitive `xxxBUNDLE` data structures that have the respective component.

There are four queries that determine the color and mix attribute as specified by GpiSetxxx functions:

- GpiQueryColor
- GpiQueryMix
- GpiQueryBackColor
- GpiQueryBackMix.

If the area color, area background color, mix attribute, or background mix attribute were specified individually, the aforementioned queries can return a value inconsistent with the current area attribute.

Area Brackets

Areas also are referred to as *area brackets*, because the functions that create and define the area always are “bracketed” by the two functions, GpiBeginArea and GpiEndArea. Only one area is defined between these two functions. However, within the area, there can be any number of adjacent, intersecting, or completely separate area primitives.

GpiBeginArea signals the start of a group of primitives that define the boundary of the area. The current position is not changed by GpiBeginArea, although it is updated by any drawing instructions that follow.

If you call GpiSetCurrentPosition or GpiMove within an area definition, the current figure is closed automatically, and a new figure is started at the current position by the next line or curve. A figure in the area, whose start and end points are not the same, is closed by drawing a straight line from the current position to the start position of the figure.

GpiEndArea signals the end of an area definition and tells the operating system to draw the area. If an application does not close a figure before issuing GpiEndArea, the figure is closed automatically by drawing a straight line from the end point of the last line or curve to the start point of the current figure. The current position is always updated to the end point of the last line drawn.

The functional sequence to draw the area in Figure 5-1 could be:

```
#include <os2.h>
void fncAREA01(void){

  #if 0

  GpiBeginArea          /* Starts the area bracket          */
  GpiSetCurrentPosition /* Set the start point of the          */
                        /* outer hexagon.                  */
  GpiPolyLine           /* Draw the outer hexagon.           */
  GpiSetCurrentPosition /* Set the start point of the inner  */
                        /* hexagon.                         */
  GpiPolyLine           /* Draw the inner hexagon.           */

  GpiEndArea            /* End the area definition.          */

  #endif

}
```

Figure 5-5. Defining an Area Primitive. An area bracket contains the functions that define an area. Only one area can be defined between “bracket” functions.

Following the beginning of an area bracket, an application can define the shape and location of the area in world space by using the following line and arc functions:

- GpiBox
- GpiFullArc
- GpiLine
- GpiPartialArc
- GpiPointArc
- GpiPolyFillet
- GpiPolyFilletSharp
- GpiPolyLine
- GpiPolySpline

Note: If an application calls either GpiBox or GpiFullArc inside an area, it must use the DRO_OUTLINE option.

In addition to the drawing functions, an application also can use the following specification functions inside an area bracket:

- GpiBeginElement
- GpiCallSegmentMatrix
- GpiComment
- GpiElement
- GpiEndArea
- GpiEndElement
- GpiGetData
- GpiLabel
- GpiMove
- GpiOffsetElementPointer
- GpiPop
- GpiPutData
- GpiSetArcParams
- GpiSetAttrMode
- GpiSetAttrs
- GpiSetColor
- GpiSetCurrentPosition
- GpiSetEditMode
- GpiSetElementPointer
- GpiSetElementPointerAtLabel
- GpiSetLineEnd
- GpiSetLineJoin
- GpiSetLineType
- GpiSetLineWidth
- GpiSetMix
- GpiSetModelTransformMatrix
- GpiSetSegmentTransformMatrix

An application also can use the following query functions inside an area bracket:

- GpiQueryArcParams
- GpiQueryAttrMode
- GpiQueryAttrs
- GpiQueryBackColor
- GpiQueryBackMix
- GpiQueryBoundaryData
- GpiQueryCharAngle
- GpiQueryCharBox
- GpiQueryCharDirection
- GpiQueryCharMode
- GpiQueryCharSet
- GpiQueryCharShear
- GpiQueryCharStringPos
- GpiQueryCharStringPosAt
- GpiQueryClipBox
- GpiQueryClipRegion
- GpiQueryColor
- GpiQueryColorData
- GpiQueryColorIndex
- GpiQueryCp
- GpiQueryCurrentPosition
- GpiQueryDefaultViewMatrix
- GpiQueryDefCharBox
- GpiQueryDevice
- GpiQueryDeviceBitmapFormats
- GpiQueryEditMode
- GpiQueryFontFileDescriptions
- GpiQueryFontMetrics
- GpiQueryFonts
- GpiQueryGraphicsField
- GpiQueryInitialSegmentAttrs
- GpiQueryKerningPairs
- GpiQueryLineEnd
- GpiQueryLineJoin
- GpiQueryLineType
- GpiQueryLineWidth
- GpiQueryLineWidthGeom
- GpiQueryLogColorTable
- GpiQueryMarker
- GpiQueryMarkerBox
- GpiQueryMarkerSet
- GpiQueryMix
- GpiQueryModelTransformMatrix
- GpiQueryNearestColor
- GpiQueryNumberSetIds
- GpiQueryPageViewport
- GpiQueryPattern
- GpiQueryPatternRefPoint
- GpiQueryPatternSet
- GpiQueryPel
- GpiQueryPickAperturePosition
- GpiQueryPickApertureSize
- GpiQueryRealColors
- GpiQueryRegionBox
- GpiQueryRegionRects
- GpiQueryRGBColor
- GpiQuerySegmentAttrs
- GpiQuerySegmentNames
- GpiQuerySegmentPriority
- GpiQuerySegmentTransformMatrix
- GpiQuerySetIds
- GpiQueryStopDraw
- GpiQueryTag
- GpiQueryViewingLimits
- GpiQueryViewingTransformMatrix
- GpiQueryWidth

Area Bracket Attributes

In addition to the attributes controlled by the AREABUNDLE data structure, there are two attributes that can be specified for each individual area bracket:

- Area boundary
- Area construction:

These bracket attributes affect how the area primitives inside the bracket are drawn.

The concept of *attribute currentness*, first described in Chapter 3, "Line and Arc Primitives," is especially important to areas and paths that contain multiple figures.

Area Boundaries

An application specifies the *area boundary* when it calls GpiBeginArea. There are two options:

- BA_BOUNDARY (the default)
- BA_NOBOUNDARY.

The BA_BOUNDARY value tells the programming interface to draw all outlines of the area primitives within the area bracket, using a line that conforms to the current LINEBUNDLE attributes. If the line attributes have not been changed, the default outline color is black on most displays and printers, and the default line style is

solid. An application can change the line color and line styles within the area bracket. The interior of the area primitive is filled with the current pattern.

To prevent the interface from outlining an area, an application can use the `BA_NOBOUNDARY` flag when issuing `GpiBeginArea`. Only the interior fill pattern is visible. This value most often is used when an area contains many overlapping figures, and the interior lines are not desired.

You could think of this option in terms similar to the outline and fill options discussed with boxes and full arcs in Chapter 3, "Line and Arc Primitives." The `BA_BOUNDARY` value corresponds to `DRO_OUTLINEFILL`; and `BA_NOBOUNDARY`, to `DRO_FILL`. There is no distinct boundary value that corresponds to `DRO_OUTLINE`, the simple outline. To simulate this effect, OR the `BA_NOBOUNDARY` value with the appropriate area construction value when issuing `GpiBeginArea`.

Area Construction

An application specifies the *area construction* when it calls `GpiBeginArea`. There are two options:

- `BA_ALTERNATE` (the default)
- `BA_WINDING`.

Construction modes also are called *area-fill* modes; they provide different ways of determining whether a given point is included in the filled area. Normally, the construction mode you use is a matter of personal preference.

Alternate Mode

In alternate mode, the following occurs:

- A point is included in the filled area if you have to cross an odd number of lines in the area when drawing a line from that point to infinity.
- A point is not included in the filled area if you have to cross an even number of lines in the area when drawing a line from that point to infinity.

In the example in Figure 5-6, the inner hexagon is not shaded, because to draw a line from any point in the hexagon to infinity, you must cross 2 boundary lines. The remainder of the area is shaded, because to move outside the area, you need to cross only 1 boundary line.

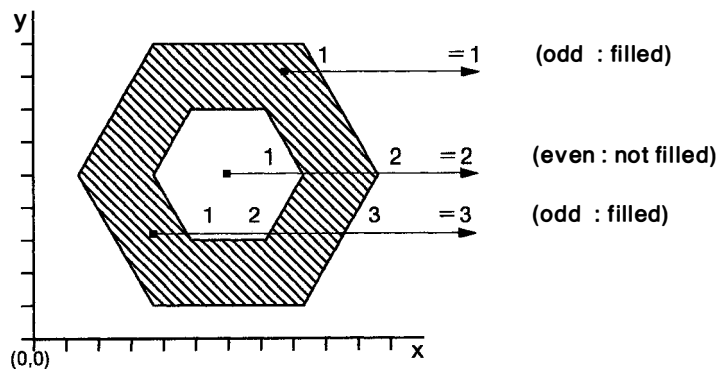


Figure 5-6. Calculating Filled Areas Constructed in Alternate Mode

Winding Mode

In winding mode, the direction in which the boundary lines in the area are drawn determines whether a given point is included in the filled area. The direction of a line depends on both the graphics functions used to draw it and the world coordinates that define it. For example, if the current position of a presentation space is (c,c) , and `GpiBox` is called with the diagonally-opposite corner of the box specified as (d,d) , `GpiBox` always draws the box in the following order:

1. (x_c, y_c) to
2. (x_d, y_c) to
3. (x_d, y_d) to
4. (x_c, y_d) and returning to
5. (x_c, y_c) .

As illustrated in Figure 5-7, in some cases the box is drawn in a counterclockwise direction. When either x_d is less than x_c , or y_d is less than y_c , but not both, a box is drawn clockwise.

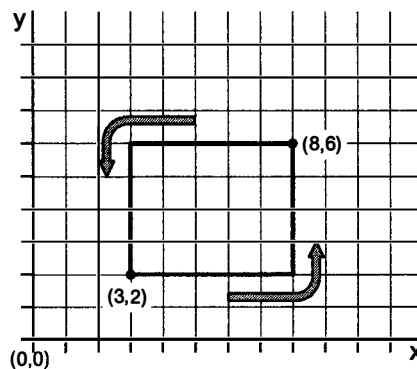


Figure 5-7. The Box. The current position is $(3,2)$ and the specified corner is at $(8,6)$. The box is drawn from $(3,2)$ to $(8,2)$ to $(8,6)$ to $(3,6)$ to $(3,2)$.

For the polyline primitive, the direction in which a line is drawn depends on the relative values of the end points of each line, so you can choose whether to draw in a clockwise or counterclockwise direction.

To determine if a given point is included in the filled area, count the number of lines to be crossed to move from that point to infinity. For each boundary line drawn in one direction, add one to the tally. For each line drawn in the opposite direction, subtract one from the tally. A point is within the area if the result is nonzero.

If two figures—for example, the hexagons in Figure 5-8—are drawn in different directions, the tally for the inner hexagon is 0 and the area will look exactly as it does in alternate mode.

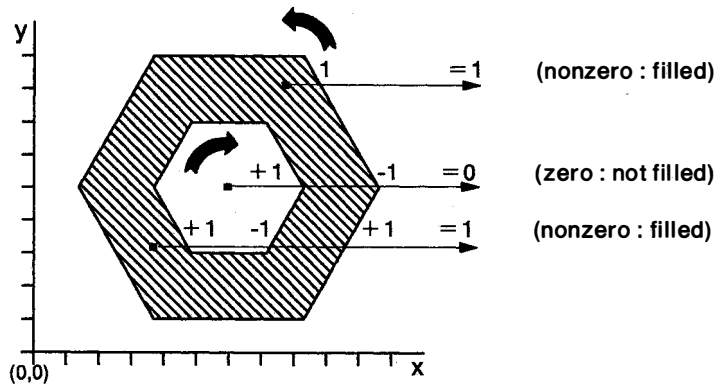


Figure 5-8. Area Constructed in Different Directions in Winding Mode

If the two hexagons are drawn in the same direction, the result is 2, and the inner hexagon is shaded as shown in Figure 5-9.

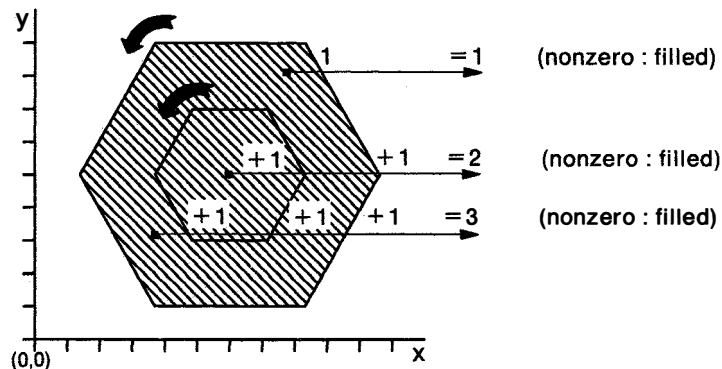


Figure 5-9. Area Constructed in the Same Direction in Winding Mode

The boundary lines of the area in Figure 5-9 have been drawn and are visible through the area-fill pattern. The boundary lines of an area primitive do not have to be drawn; but if they are, they are drawn according to the current line attributes.

To vary the appearance of different parts of the boundary line, you can change the current line attributes during area definition.

The boundary lines of an area are considered a part of the area's interior. Therefore, when you draw an area without boundary lines (BA_NOBOUNDARY), the area-fill pattern extends to include the boundaries of the area, and the area is the same size it would be if the boundary lines had been drawn.

Attribute Currentness

Graphics functions that change attributes are valid within the area bracket. There are misconceptions, however, about which attributes are used when the figures are displayed in a window or drawn to the printer.

Review the following functional sequence:

Function	Effect
GpiColor	Sets color to red.
GpiBeginArea	Opens area bracket.

GpiLine	Draws a straight line.
GpiColor	Sets color to green.
GpiPolyLine	Draws two additional lines to form a triangle.
GpiEndArea	Ends the area bracket and displays a picture.

If you are displaying on a color monitor, the image appears as follows:

The triangle has one red line, drawn with GpiLine, and two green lines, drawn with GpiPolyLine. The interior of the triangle, if drawn, is red. Red is the color that was current when the area was defined and, therefore, is the current color for the area interior. In terms of the interior area, green is only the color that was current when the function that initiates drawing was called.

If the application immediately opened a new area bracket, the interior of that bracket, if drawn, would be green. The color attribute for the second AREABUNDLE, is not "inherited" from the last AREABUNDLE drawn.

About Polygon Primitives

Area brackets have the flexibility to draw any combination of curved or straight lines and combine the results into a closed figure. An advantage that polygons sometimes have over area brackets, however, is that they require fewer time-consuming calculations to provide that kind of flexibility.

The operating system provides a function that enables you to draw multiple straight-line closed areas outside of an area bracket; and like bracket-generated areas, they can be adjacent, intersecting, or completely separate.

The function is GpiPolygons, and the set of objects it draws are called *polygon primitives*. A polygon primitive is any set of polygons with specified vertices that can be filled or filled and outlined. This function accepts as input the desired number of polygons, the POLYGON data structure for each polygon, and a boundary and construction option similar to those of the GpiBeginArea function.

The purpose of GpiPolygons is to enable you to specify an area in such a way as to pass all of the area boundaries at once. This improves performance significantly because the operating system cannot process the accumulated primitives inside an area bracket until it reaches the GpiEndArea function. GpiPolygons is not valid inside an area bracket because the figures it defines are areas already.

Although GpiPolygons is limited to straight line figures, it retains a great deal of flexibility. The polygon data structure (POLYGON) contains the number of vertices; and the vertices, themselves, are in world coordinates. The collection of POLYGON structures accepted in a single GpiPolygons function is not limited to one type of polygon.

The drawing of the first polygon begins at the current position. For all subsequent polygons, all vertices must be explicitly defined. If the individual polygons are not completely defined, they are closed with a straight line drawn from the last defined vertex to the first.

After the application has defined the polygons, they may be transformed and manipulated just as other primitives are.

Polygon Boundaries

Applications have two options when specifying the polygon boundary:

- `POLYGON_BOUNDARY` (the default)
- `POLYGON_NOBOUNDARY`.

The `POLYGON_BOUNDARY` value tells the PM programming interface to draw all outlines of the polygon primitives using a line that conforms to the current `LINEBUNDLE` attributes. If the line attributes have not been changed, the default outline color is black on most display devices and printers, and the default line style is solid. As the attributes cannot be changed within the context of `GpiPolygons`, all polygons are drawn with the same line. The interior of the polygons are filled with the pattern that conforms to the current `AREABUNDLE` attributes.

To prevent PM from outlining an area, an application can use the `POLYGON_NOBOUNDARY` flag when issuing `GpiPolygons`. Only the interior fill pattern is visible. This value is used most often when there are many overlapping polygons, and the interior lines are not desired.

You might think of this option in terms similar to the outline and fill options discussed with boxes and full arcs in Chapter 3, "Line and Arc Primitives." The `POLYGON_BOUNDARY` value corresponds to the `DRO_OUTLINEFILL` value; and `POLYGON_NOBOUNDARY`, to the `DRO_FILL`. There is no distinct boundary value that corresponds to `DRO_OUTLINE`, the simple outline.

To simulate this effect, OR the `POLYGON_NOBOUNDARY` value with the appropriate polygon construction value, described below, when issuing `GpiPolygons`.

Polygon Construction

An application specifies the area construction when it calls `GpiBeginArea`. There are two options:

- `POLYGON_ALTERNATE` (the default)
- `POLYGON_WINDING`.

As with area construction, in alternate mode:

- A point is included in the filled polygon if you have to cross an odd number of lines in the set of polygons when drawing a line from that point to infinity.
- A point is not included in the filled polygon if you have to cross an even number of lines in the set of polygons when drawing a line from that point to infinity.

Also as with area construction, in winding mode, the direction in which the boundary lines of the polygons are drawn determines whether a given point is included in the filled polygon. Since the individual polygons drawn with `GpiPolygons` are generated by independent structures, the direction in which a polygon is drawn depends only on the vertices of that polygon.

To determine if a given point is included in the filled polygon, count the number of lines to be crossed to move from that point to infinity. For each boundary line drawn in one direction add one to the tally. For each line drawn in the opposite direction, subtract one from the tally. A point is within the polygon if the result is nonzero.

Polygon Overlap

An application specifies which pels are drawn when it calls `GpiPolygons`. There are two options:

- `POLYGON_INCL` (the default)
- `POLYGON_EXCL`.

When the overlap value is `POLYGON_INCL`, the bottom right is included in the polygon. The value `POLYGON_EXCL`, the exclusive value, indicates that the bottom right is excluded from the polygon. This value acts in conjunction with the mix attribute in determining the appearance of a polygon, which is especially important for a group of overlapping or adjacent polygons.

For example, `GpiPolygons` specifies a number of polygons. Two of the polygons share the vertex pair (6,7) and (9,7). When the polygons are drawn, if the overlap value was `POLYGON_INCL`, there are two distinct lines to be drawn from (6,7) to (9,7). A mix attribute other than `FM_OVERPAINT` could cause undesirable results as the line and drawing-surface colors mix.

As with the polygon boundary and construction values, the overlap value can be ORed when issuing `GpiPolygons` to create a specific effect.

Using Area and Polygon Primitives

You can use area functions to:

- Draw one or more closed figures
- Create a custom fill pattern from a bit map
- Create a custom fill pattern from a font character.

Drawing a Single, Closed Figure

Figure 5-10 shows an example of how to use area functions to draw a single closed figure that is filled with a vertical pattern using the alternate mode. The closed figure in this example is a 5-pointed star.

```
#define INCL_GPI
#include <os2.h>
void fncAREA02(void){
    POINTL aptl[5];          /* Structure for current position */
    HPS hps;

    /* Initialize the array of points for the 5-pointed star. */
    aptl[0].x = 400; aptl[0].y = 195;
    aptl[1].x = 40; aptl[1].y = 320;
    aptl[2].x = 260; aptl[2].y = 10;
    aptl[3].x = 260; aptl[3].y = 390;
    aptl[4].x = 37; aptl[4].y = 82;

    GpiSetPattern(hps, PATSYM_VERT); /* Set pattern outside bracket */

    /* Draw the star. */
    GpiBeginArea(hps, BA_ALTERNATE);
    GpiMove(hps, &aptl[4]);          /* First and last point of star */
    GpiPolyLine(hps, 5L, aptl);
    GpiEndArea(hps);
} /* fncAREA02 */
```

Figure 5-10. Drawing a Closed Figure

Drawing Multiple, Intersecting, Closed Figures

Figure 5-11 is an example of how to use area functions to draw two intersecting boxes, filled, using the winding mode.

```
#define INCL_GPI
#include <os2.h>
void fncAREA03(void){
    POINTL ptl;          /* Structure for current position */
    HPS hps;

    GpiBeginArea(hps, BA_WINDING);
    ptl.x = 100;
    ptl.y = 50;
    GpiMove(hps, &ptl);
    ptl.x = 300;
    ptl.y = 250;
    GpiBox(hps, DRO_OUTLINE, &ptl, 0, 0);
    ptl.x = 180;
    ptl.y = 120;
    GpiMove(hps, &ptl);
    ptl.x = 380;
    ptl.y = 320;
    GpiBox(hps, DRO_OUTLINE, &ptl, 0, 0);
    GpiEndArea(hps);
} /* fncAREA03 */
```

Figure 5-11. Drawing Multiple Intersecting Closed Figures

Creating a Custom Fill Pattern from a Bit Map

Figure 5-12 is an example of how to create a custom fill pattern by using a hard-coded bit map. In this example, the bit map creates a pattern of arrows.

```

#define INCL_DOS
#define INCL_GPI
#define INCL_WIN
#include <os2.h>
LONG lcidCustom;          /* Bit map tag */
HPS hps;

VOID CreatePattern(VOID);

VOID MyFunction(VOID){
    CreatePattern();
    GpiSetPatternSet(hps, lcidCustom);
    .
    .
} /* func */

VOID CreatePattern(VOID){
    HBITMAP hbm;          /* Bit map handle */
    BITMAPINFOHEADER2 bmp2; /* Structure for bit map information */
    PBITMAPINFO2 pbmi2;    /* Pointer to structure for bit map data */
    PRGB2 prgb2;          /* Structure for color data */
    ULONG cbBitmapInfo, cColors;

    BYTE abPattern[] = { 0xFF, 0xFF, 0xE7, 0xFF,
                        0xE7, 0xFF, 0xC3, 0xFF,
                        0xC3, 0xFF, 0x81, 0xFF,
                        0x81, 0xFF, 0xE7, 0xFF,
                        0xE7, 0xFF, 0xE7, 0xFF,
                        0xE7, 0xFF, 0xE7, 0xFF,
                        0xE7, 0xFF, 0xE7, 0xFF,
                        0xE7, 0xFF, 0xFF, 0xFF };

    lcidCustom = 1;          /* Bit map tag */

    bmp2.cbFix = (ULONG) sizeof(BITMAPINFOHEADER2);
    bmp2.cx = 8;             /* Bit map is 8 pels wide */
    bmp2.cy = 8;             /* Bit map is 8 pels high */
    bmp2.cPlanes = 1;        /* One bit plane */
    bmp2.cbBitCount = 1;     /* One bit per pel */

    /* Use default values for the remainder of the structure. */

    bmp2.ulCompression = 0;
    bmp2.cbImage = 0;
    bmp2.cxResolution = 0;
    bmp2.cyResolution = 0;
    bmp2.cclrUsed = 0;
    bmp2.cclrImportant = 0;
    bmp2.usUnits = 0;
    bmp2.usReserved = 0;
    bmp2.usRecording = 0;
    bmp2.usRendering = 0;
    bmp2.cSize1 = 0;
    bmp2.cSize2 = 0;
    bmp2.ulColorEncoding = 0;
    bmp2.ulIdentifier = 0;

```

Figure 5-12 (Part 1 of 2). Creating a Custom Fill Pattern from a Bit Map


```

cColors = 1 << (bmp2.cBitCount * bmp2.cPlanes);

cbBitmapInfo = sizeof(BITMAPINFO2) + (sizeof(RGB2) * cColors);

DosAllocMem((PVOID)&pbmi2, cbBitmapInfo,
            PAG_COMMIT | PAG_READ | PAG_WRITE);

pbmi2->cbFix = bmp2.cbFix;
pbmi2->cx = bmp2.cx;
pbmi2->cy = bmp2.cy;
pbmi2->cPlanes = bmp2.cPlanes;
pbmi2->cBitCount = bmp2.cBitCount;

/* Use default values for the remainder of the structure. */

pbmi2->ulCompression = 0;
pbmi2->cbImage = 0;
pbmi2->cxResolution = 0;
pbmi2->cyResolution = 0;
pbmi2->cclrUsed = 0;
pbmi2->cclrImportant = 0;
pbmi2->usUnits = 0;
pbmi2->usReserved = 0;
pbmi2->usRecording = 0;
pbmi2->usRendering = 0;
pbmi2->cSize1 = 0;
pbmi2->cSize2 = 0;
pbmi2->ulColorEncoding = 0;
pbmi2->ulIdentifier = 0;

prgb2 = (PRGB2) (pbmi2 + 1); /* Set address to follow bmp2 */

/* Set bit map colors to black and white. */
prgb2[0].bBlue = 0; /* Color[0] = black */
prgb2[0].bGreen = 0; /* Color[0] = black */
prgb2[0].bRed = 0; /* Color[0] = black */
prgb2[0].fcOptions = 0;
prgb2[1].bBlue = 255; /* Color[1] = white */
prgb2[1].bGreen = 255; /* Color[1] = white */
prgb2[1].bRed = 255; /* Color[1] = white */
prgb2[1].fcOptions = 0;

/* Create a bit map and retrieve its handle. */
hbm = GpiCreateBitmap(hps,
                    &bmp2,
                    CBM_INIT,
                    (PBYTE) abPattern, /* Array of bits */
                    pbmi2);

/* Tag the bit map just created with a custom identifier (lcid). */
GpiSetBitmapId(hps, hbm, lcidCustom);
} /* CreatePattern */

```

Figure 5-12 (Part 2 of 2). Creating a Custom Fill Pattern from a Bit Map

Creating a Custom Fill Pattern from a Font Character

Figure 5-13 is an example of how to create a custom fill pattern by using a character from a font. The fill pattern can be only an 8-by-8 bit map; as a result, not all of the character is used.

```
#define INCL_GPI
#define INCL_WIN
#include <os2.h>

HPS hps; /* Presentation-space handle */
LONG lcidCustom; /* Font identifier */
FONTMETRICS afm[80];
FATTRS fat;

VOID LoadFont(VOID);

void fncAREA05(void){
    figseg fit=11.
        LoadFont();
    GpiSetPatternSet(hps, lcidCustom);
    GpiSetPattern(hps, 'o'); /* Use lowercase 'o' as fill */
    .
    .
    .
} /* fncAREA05 */

VOID LoadFont(VOID){
    LONG cFonts = 0;
    LONG cPublicFonts, i;

    lcidCustom = 1;

    /* Determine the number of loaded public fonts. */
    cPublicFonts = GpiQueryFonts(hps, QF_PUBLIC, NULL, (PLONG) &cFonts,
        (LONG) (sizeof(FONTMETRICS)), NULL);

    /* Load the metrics for all public fonts into afm. */
    GpiQueryFonts(hps, QF_PUBLIC, NULL, (PLONG) &cPublicFonts,
        (LONG) (sizeof(FONTMETRICS)), afm);

    /* Get the first image font with a point size larger than 8. */
    for (i = 0; ((afm[i].fsDefn & FM_DEFN_OUTLINE) ||
        afm[i].lEmHeight <= 8) && i < cPublicFonts; i++);

    /* Load the FATTRS structure with the required metrics. */
    fat.usRecordLength = sizeof(fat);
    fat.fsSelection = 0;
    fat.lMatch = afm[i].lMatch;
    StringCopy(fat.szFacename, afm[i].szFacename);
    fat.idRegistry = 0;
    fat.usCodePage = 0;
    fat.lMaxBaselineExt = 0;
    fat.lAveCharWidth = 0;
    fat.fsType = 0;
    fat.fsFontUse = 0;

    /* Select this font and assign it a custom lcid. */
    GpiCreateLogFont(hps, NULL, lcidCustom, &fat);
    GpiSetCharSet(hps, lcidCustom);
} /* LoadFont */
```

Figure 5-13. Creating a Custom Fill Pattern from a Font Character

Summary

Table 5-3 summarizes the area and polygon primitive functions.

<i>Table 5-3. Area-Primitive Functions</i>	
Function Name	Description
GpiBeginArea	Starts the area bracket and sets the outline option and fill mode.
GpiEndArea	Ends the area bracket and draws the area.
GpiPolygons	Draws a series of closed, straight-line figures.
GpiQueryPattern	Determines the current bit map fill pattern.
GpiQueryPatternRefPoint	Determines the fill-pattern reference point in world coordinates.
GpiQueryPatternSet	Determines the local identifier of a bit map or font.
GpiSetPattern	Defines the fill pattern from the supplied patterns or from a code point for a character or symbol in a font.
GpiSetPatternRefPoint	Defines the fill-pattern reference point in world coordinates.
GpiSetPatternSet	Defines the fill-pattern attribute to a local identifier of a bit map or font.

Table 5-4 summarizes the data structures used by the area and polygon primitive functions.

<i>Table 5-4. Area-Primitive Structure</i>	
Structure Name	Description
AREABUNDLE	A structure of area primitive attributes.
LINEBUNDLE	A structure of line primitive attributes, which controls the outline of the area primitives.
POINTL	A structure specifying the values a single pair of x- and y-coordinates.
POLYGON	A structure that describes a straight line polygon. It contains a count of the vertices, and an array of POINTL structures to describe the coordinates of each vertex.

Chapter 6. Character String Primitives

Character string primitives are printed or displayed text limited to a length of 256 characters by the PM programming interface.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Color and mix attributes
- Fonts
- Coordinate spaces
- Transformations.

About Character String Primitives

A PM application must make certain choices about fonts in preparation for text display or printing. Often these choices are driven by selections from a user. After the selections are made, as discussed in Chapter 9, "Fonts," the application can make additional decisions about the appearance of the individual characters within the font by setting attributes in the CHARBUNDLE data structure. CHARBUNDLE is the lowest of three levels of data structures that define the appearance of displayed or printed text. The other two, FONTMETRICS and FATTRS, are described in Chapter 9, "Fonts."

A font family—for instance, Helvetica[®] or Courier—is a collection of fonts. Fonts within the same family share certain attributes such as stroke width and serif characteristics. However, the individual fonts differ from each other in the following ways:

- Height
- Line weight
- Appearance.

Stroke width refers to the width of lines used to draw characters and symbols from a font. A *serif* is a short crossline drawn at the ends of the main strokes that form a character or symbol. Height refers to the point size of a font. A common example of line weight is **boldface**. A common example of appearance is *italic*.

The most important factor that affects the CHARBUNDLE attributes is whether the current font is an *image* or an *outline* font. Figure 6-1 on page 6-2 shows an example of the difference.

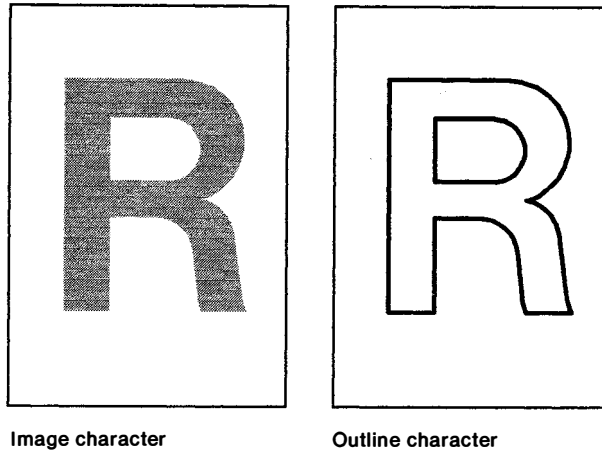


Figure 6-1. Image and Outline Fonts

Outline fonts are composed of characters drawn with straight and curved lines. Image fonts, also called bit map fonts, are composed of pels arranged in certain shapes.

Attributes of Character String Primitives

Attributes of the character string primitives contained in CHARBUNDLE are as follows:

- Character set
- Character mode
- Character cell
- Character angle
- Character shear
- Character direction
- Character text alignment
- Character extra
- Character break extra
- Foreground color
- Background color
- Foreground mix
- Background mix.

When an application creates a presentation space, the character string attributes are set to the default values shown in Table 6-1.

<i>Table 6-1 (Page 1 of 2). Character String Attribute Default Values</i>		
Attribute	Default Value	Function that Redefines Attribute
Set	LCID_DEFAULT	GpiSetCharSet
Mode	CM_MODE1	GpiSetCharMode
Cell	• Outline font – Defined by device • Image font – Defined by font	GpiSetCharBox
Angle	(1,0)	GpiSetCharAngle
Shear	(0,1)	GpiSetCharShear
Direction	Left to right	GpiSetCharDirection

Table 6-1 (Page 2 of 2). Character String Attribute Default Values

Attribute	Default Value	Function that Redefines Attribute
Text alignment	Left	GpiSetTextAlignment
Extra	0	GpiSetCharExtra
Break extra	0	GpiSetCharBreakExtra
Foreground color	Black	GpiSetAttrs (CBB_COLOR)
Background color	Clear	GpiSetAttrs (CBB_BACK_COLOR)
Foreground mix	Overpaint	GpiSetAttrs (CBB_MIX_MODE)
Background mix	Leave alone	GpiSetAttrs (CBB_BACK_MIX_MODE)

Current character attributes take effect when you send graphics characters to an output device; they have no effect at the time you define a logical font.

Character Set

Character set defines the local identifier, *lcid*, of the current font. The font *lcid* is set using GpiSetCharSet. GpiCreateLogicalFont, described in Chapter 9, "Fonts," associates the font with an *lcid*.

The value of the current *lcid* is determined using GpiQueryCharSet.

Character Mode

Every presentation space has a current *character mode*, which determines the extent to which a font can be affected by the attributes defined in CHARBUNDLE. The mode affects compatibility issues that are invisible to both the user and the programmer. Table 6-2 describes the character modes and their influences on the current font.

An application selects a character mode using GpiSetCharMode. When a mode is selected, it applies to any font (including the system font) used while the mode is current. The character mode takes effect when you draw character strings; it has no effect at the time you define a logical font.

Table 6-2. Character Mode Effects on Character Attributes		
Character Mode	If Image Font...	If Outline Font...
CM_MODE1	Ignores all current character attributes, except character direction.	Affected by all current character attributes.
CM_MODE2	Uses the character shear, box, angle, and direction attributes to position the character cell, but the characters themselves are not sheared, rotated, scaled, or reversed.	Affected by all current character attributes.
CM_MODE3	Raises an error if you try to draw a character string. Note: Any font used when the current character mode is CM_MODE3 <i>must</i> be an outline font.	Affected by all current character attributes.

The default character mode (CM_DEFAULT) is identical to CM_MODE1.

Character Cell

A *character cell* is an imaginary rectangular boundary that defines the horizontal and vertical space occupied by a single character from an outline character set.

The PM programming interface calculates character cell width and height from the point size. The width value for the character cell is the nominal width of the lowercase characters in the character set. In a *monospace* font, the width of all character cells is identical. In a *proportional* font, the width of the character cells depends on the character.

In an image font, the height of the character cell is the number of pels in the font. In an outline font, the height of the character cell is the point size of the font.

The characters in a character string are positioned one character per cell. The spacing between adjacent characters in a string is affected by the character cell attribute, except for image characters in CM_MODE1.

Cell width determines the spacing of consecutive characters along the baseline, as illustrated in Figure 6-2 on page 6-5.

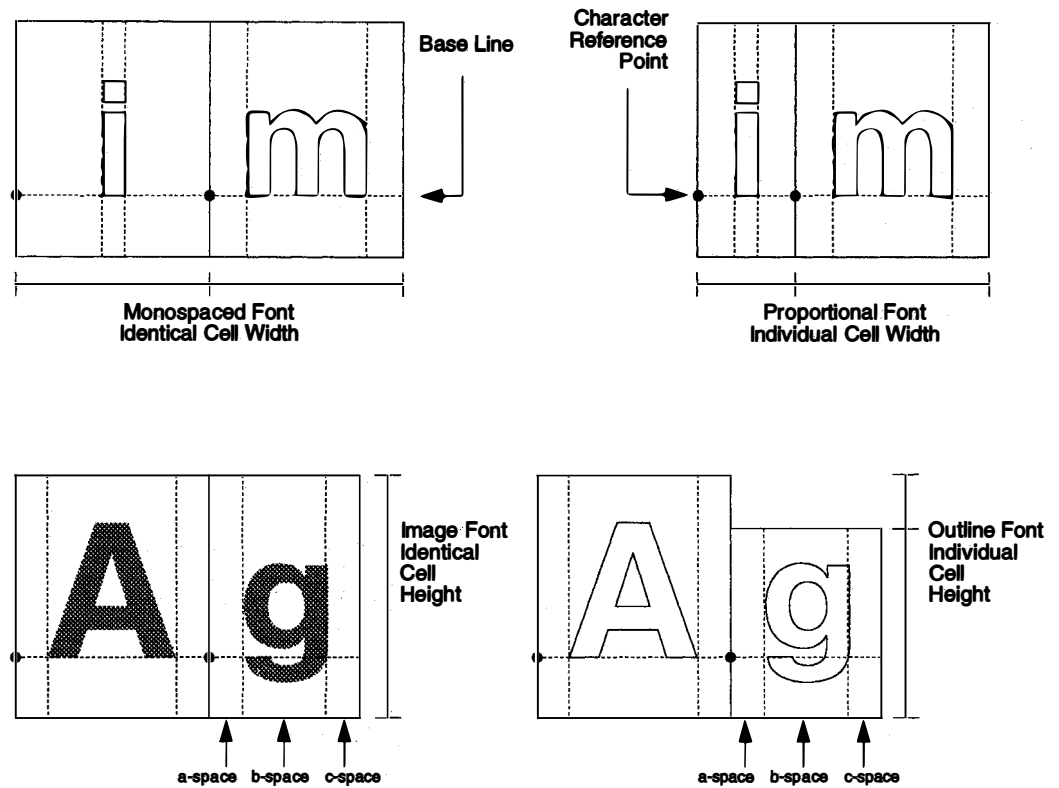


Figure 6-2. Character Cell Measurements

Current cell size is specified using `GpiSetCharBox`. As input, this function accepts the desired height and width of the character cell in world coordinates. These values are related to certain dimensions in the `FONTMETRICS` structure that controls font attributes. Heights or widths of 0 are valid input and cause the outline character to be drawn as a point or straight line. Heights or widths of negative values cause certain special effects, for example, reversed lettering.

The character cell value affects both the size and position of characters drawn from an outline font, regardless of the current character mode. Each character is scaled up or down to fit the cell size, as shown in Figure 6-3 on page 6-6.

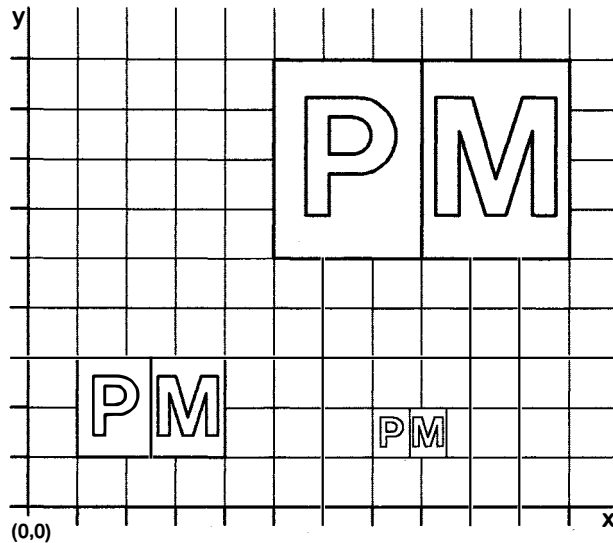


Figure 6-3. Effect of the Character Cell on an Outline Font

The character cell value is ignored if the current font is an image font and the current character mode is CM_MODE1, as shown in Figure 6-4.

Note: It is essential to code the character cell correctly, even if you anticipate using image fonts. In case of a font match failure, an outline font can be substituted for a image font.

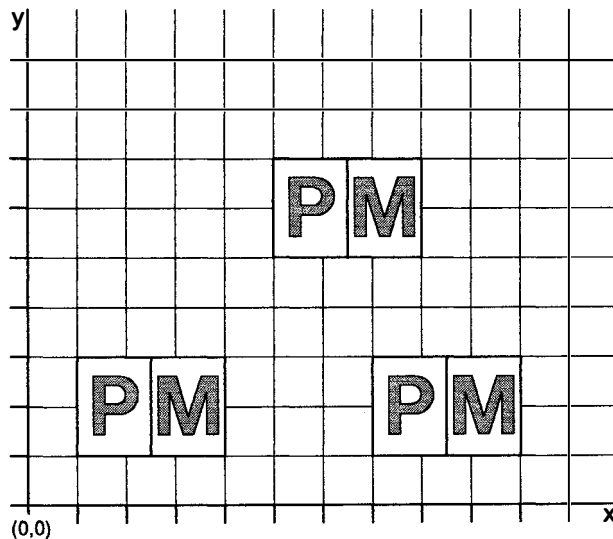


Figure 6-4. Effect of the Character Cell on an Image Font in CM_MODE1. Although the character cell has been both increased and decreased, the character string is unaltered.

The character cell value controls the *positioning* of image-font characters when the current character mode is CM_MODE2; but it cannot cause the characters to be scaled to fit the cell. This effect is shown in Figure 6-5 on page 6-7.

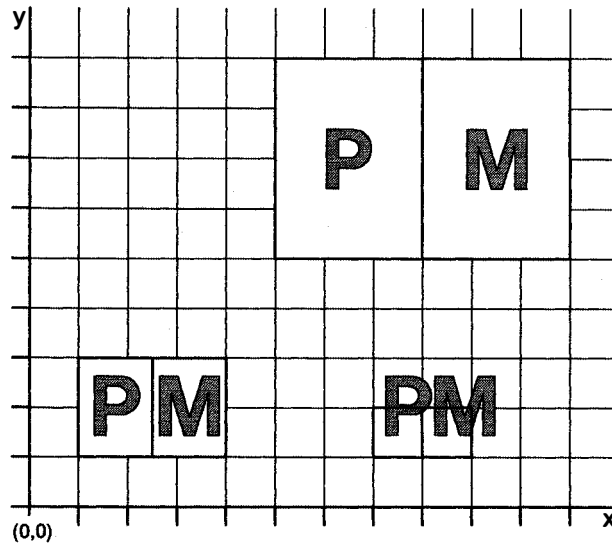


Figure 6-5. Effect of the Character Cell on an Image Font in CM_MODE2. If you increase the character cell size for an image font in CM_MODE2, the characters are more widely spaced, but their size is not changed. If you decrease character cell size, the space between the characters is reduced, and because the characters themselves cannot be scaled, they can overlap.

Default Character Cell

The default character-cell size is a device-dependent value. You do not need to know this value unless you want to switch back to the initial default value after having specified another value. It is recommended that you save initial default values using GpiSavePS.

The default character cell, like other default attributes, can be set using GpiSetDefAttrs (CBB_BOX). If necessary, you can query the current default value using GpiQueryDefAttrs.

Coding a Character Cell

The dimensions that an application passes to GpiSetCharBox and that GpiQueryCharBox returns are fixed-point values. A *fixed* value is a 32-bit value whose high-order 16 bits contain the integral part of the floating-point number and whose low-order 16 bits contain the fractional part. The fractional part is the numerator of a fraction whose denominator is fixed at 65536. The MAKEFIXED macro provides a method for producing fixed values. If, for example, one of the character cell dimensions were 7.635 world units, an application could obtain the corresponding fixed value by using the MAKEFIXED macro, passing 7 as the first argument and 41615 as the second.

The purpose of the character cell is to assist other font metrics, described in Chapter 9, "Fonts," to define text lines. For example, if you planned to have an average text line of 60 characters, dividing your output width by 60 would provide your character cell width.

DevQueryCaps can be used to provide information about suitable character cell values. DevQueryCaps returns two sets of values that can influence the character cell:

- **Default cell values**
 - CAPS_GRAPHICS_CHAR_WIDTH
 - CAPS_GRAPHICS_CHAR_HEIGHT.
- **Device resolution**
 - CAPS_HORIZONTAL_FONT_RES
 - CAPS_VERTICAL_FONT_RES.

The default cell values return the size of the default character cell in pels. Convert the device resolution into character cell values by multiplying it by the desired point size, then dividing by 72 (72.2818).

Character Angle

The *character angle* is defined by the x-axis and a vector drawn through the origin to a specified point in a Cartesian coordinate system. Neither (0,0) nor the specified point need have any relation to the current position. The operating system then aligns the baseline with this vector.

An application can retrieve the point that defines the character angle vector using GpiQueryCharAngle. An application can set the character angle using GpiSetCharAngle. This function accepts as input the x- and y-coordinates of a point that defines the new vector. When you specify an angle, it becomes the current setting. To reset the character angle vector to its default value (parallel to the x-axis), call GpiSetCharAngle with both x and y equal to 0.

The effects of the current character angle vary depending on the current character mode and the current font type. When the current font is an outline font, the current character angle determines the angle of both the whole string and the individual characters in the string, regardless of the current character mode. The baseline of each character cell is drawn parallel to the new baseline, as shown in Figure 6-6.

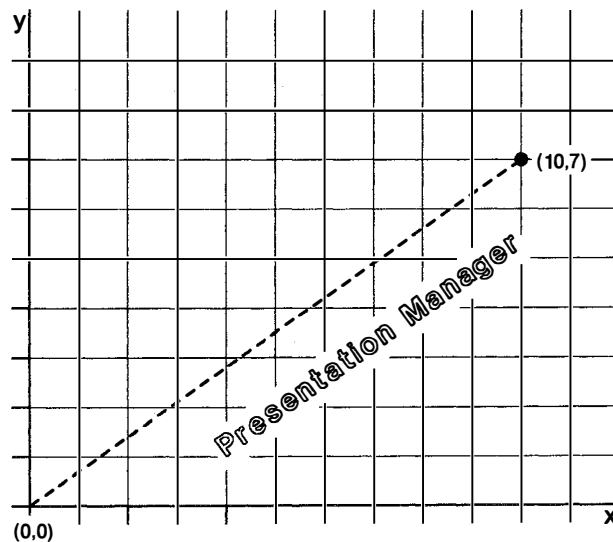


Figure 6-6. Effect of the Character Angle on an Outline Font. The character string is drawn parallel to the vector drawn from the origin to (10,7). The baseline of each of the character cells also is drawn parallel to this vector.

The character angle value is ignored if the current font is an image font and the current character mode is `CM_MODE1`, as shown in Figure 6-7.

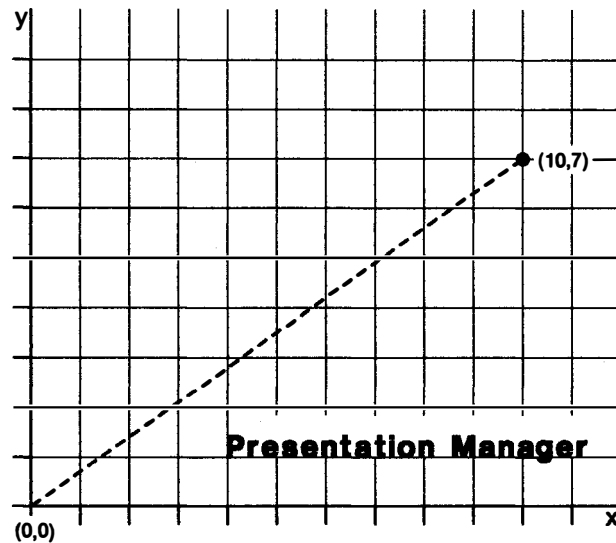


Figure 6-7. Effect of the Character Angle on an Image Font in `CM_MODE1`. The angle of the character string is unaltered by `GpiSetCharAngle`.

When the current font is an image font and the current character mode is `CM_MODE2`, the current character angle determines the angle of the whole string but does not affect the individual characters in the string. The *character reference point*, which is the point at which the character baseline intersects the left edge of the character cell, is placed on a line parallel to the new baseline. The character reference point is illustrated in Figure 6-2 on page 6-5. The baseline of each character cell remains parallel to the x-axis, as shown in Figure 6-8.

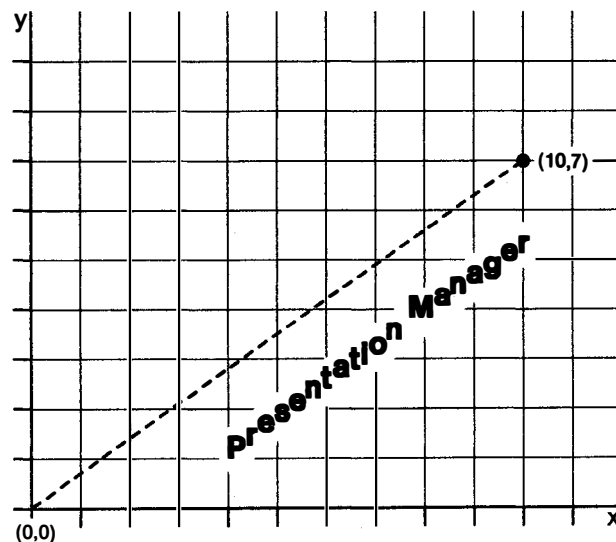


Figure 6-8. Effect of the Character Angle on an Image Font in `CM_MODE2`. The character string is drawn parallel to the vector from (0,0) to (10,7).

Each of the characters in the string in Figure 6-8 is drawn with the vertical sides of its character cell parallel to the y-axis, and with the horizontal sides of its character box parallel to the x-axis.

Character Shear

Character shear is the angle defined by the x-axis and a vector drawn through the origin to a specified point in a Cartesian coordinate system. Neither (0,0) nor the specified point need have any relation to the current position. The operating system then aligns the vertical sides of the character cell. If the font is an outline font, the operating system also aligns the vertical strokes of the characters with the vector, regardless of the current character mode, as shown in Figure 6-9.

An application can retrieve the point that defines the character shear vector using `GpiQueryCharShear`. An application can set the character shear using `GpiSetCharShear`, which accepts as input the x- and y-coordinates of a point that defines the new vector in a `POINTL` structure.

The *shear* of a character is the angle formed by the vertical lines of the character cell, and it can affect both the positioning and shape of characters in the character string. By default, the vertical lines of a character cell are parallel to the y-axis of the presentation page. As input to `GpiCharShear`, supply the coordinates of the end point of a vector drawn from the origin (0,0). The effects of the current character shear value vary, depending on the current character mode and font type.

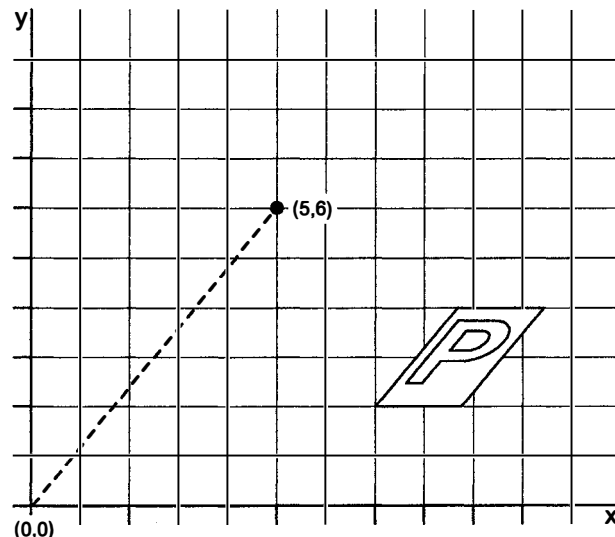


Figure 6-9. *Effect of Character Shear on an Outline Font. The character cell is drawn with its vertical lines parallel to the vector from the origin to (5,6). The character is sheared to the same degree.*

The character-shear value is ignored if the current font is an image font and the current character mode is `CM_MODE1`, as shown in Figure 6-10 on page 6-11.

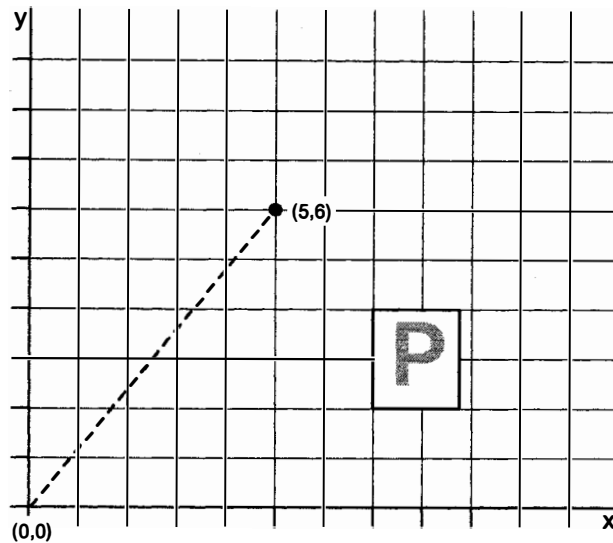


Figure 6-10. Effect of Character Shear on an Image Font in CM_MODE1. The character string is unaffected by the character-shear value.

The character-shear value affects the positioning of the characters from an image font in CM_MODE2 only if character direction is CHDIRN_TOPBOTTOM or CHDIRN_BOTTOMTOP. That is, characters drawn vertically do not appear in a vertical line for nonzero shear angle from the vertical. The characters themselves cannot be sheared, as shown in Figure 6-11.

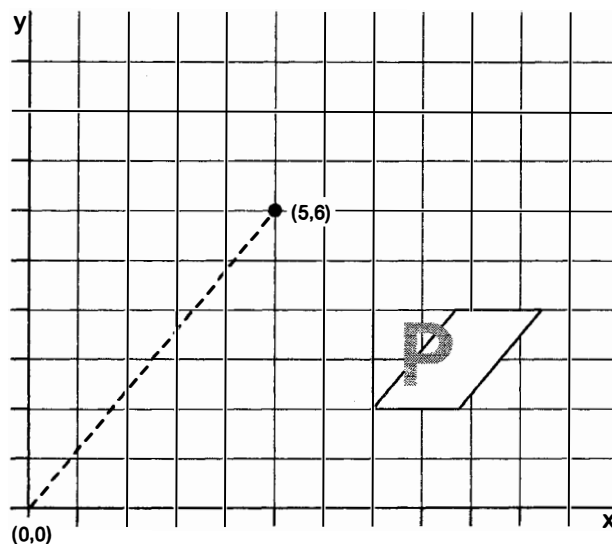


Figure 6-11. Effect of Character Shear on an Image Font in CM_MODE2. The character cell is sheared; and it controls the positioning of the characters, but the characters themselves are unchanged.

If the x- and y-coordinate values you specify in GpiSetCharShear are both positive and negative, the characters slant from lower left to upper right. If you supply one negative and one positive value, the characters slant from upper left to lower right, as illustrated in Figure 6-12 on page 6-12.

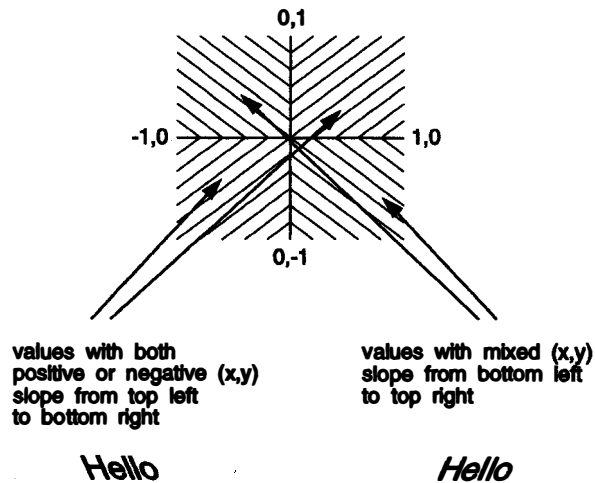


Figure 6-12. Effect of X- and Y-Values on Character Shear

Character shear, like other attributes, has a default value that can be changed using `GpiSetDefAttrs (CBB_SHEAR)`. To reset the character shear to its default effect of drawing the vertical lines of the character cell parallel to the y-axis, supply a coordinate value of (0,1) on `GpiSetCharShear`.

Character Direction

Character direction is the direction in which the characters in a string are drawn in relation to the baseline. By default, the characters are drawn from left to right. An application can change the direction using `GpiSetCharDirection`. `GpiSetCharDirection` accepts as input 1 of the 4 values illustrated in Figure 6-13. The value `CHDIRN_DEFAULT` is identical to `CHDIRN_LEFTRIGHT`.

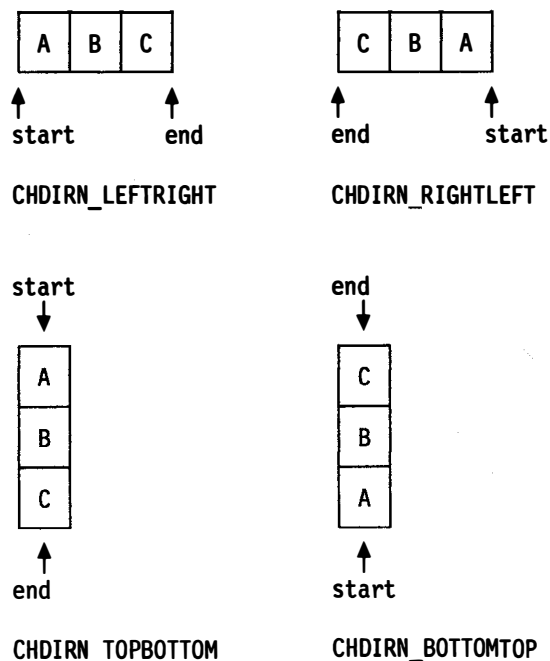


Figure 6-13. Effect of Character Direction on an Image Font or Outline Font

An application can determine the current character direction using `GpiQueryCharDirection`.

Character Text Alignment

Character text alignment is the attribute that describes how the character strings are drawn with respect to the boundary of the output, either the current position or the starting position of the string, if a `GpiCharStringxxxAt` function draws the string. The alignment is set using `GpiSetTextAlignment`, which accepts as input a long value for horizontal and vertical alignment.

The acceptable values for `GpiSetTextAlignment` depend on the direction of the current coordinate system, as follows:

Value	Corresponds to the direction of...
Left	The lowest x-coordinate value
Right	The highest x-coordinate value
Top	The highest y-coordinate value
Bottom	The lowest y-coordinate value

Internally, the PM programming interface determines a reference point within a character string that is to be positioned over the starting point specified for the string.

If an application draws the string with either `GpiCharString` or `GpiCharStringPos`, the starting point specified for the string is the current position. If the application draws the string with either `GpiCharStringAt` or `GpiCharStringPosAt`, the starting point specified for the string is accepted as input by the function.

Horizontal Alignment of a Character String

When a horizontal character string does not fill the width of the output area, it can be positioned in 1 of the 3 ways shown in Figure 6-14. All of these options can be set directly with the **IHorizontal** option of `GpiSetTextAlignment`.

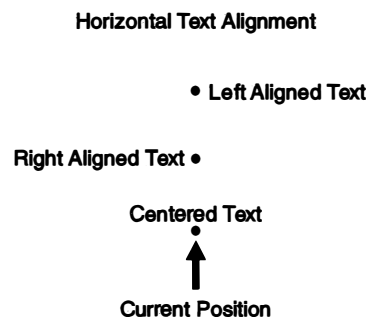


Figure 6-14. Horizontal Positioning of Text Strings

Text justification requires an application to perform both queries and coordinate calculations. The following flags are used to specify types of horizontal alignment:

Table 6-3. Horizontal Alignment Values	
Identifier	Alignment
TA_LEFT	On the left edge of the leftmost character in the string
TA_RIGHT	On the right edge of the rightmost character in the string
TA_CENTER	On the arithmetic mean of the leftmost and rightmost characters in the string

The *Presentation Manager Programming Reference* describes 2 sets of default values for the **IHorizontal** option. They are provided for compatibility and map into the horizontal alignment values described above.

GpiQueryGraphicsField, GpiQueryPageViewport, and GpiQueryViewingLimit all provide methods of determining the width of the output area so your application can specify coordinates properly for the current position.

Vertical Alignment of a Character String

When a character string is to be displayed vertically, it can be positioned in 1 of the 4 ways illustrated in Figure 6-15. The vertical options all can be set directly using the **IVertical** option of GpiSetTextAlignment.

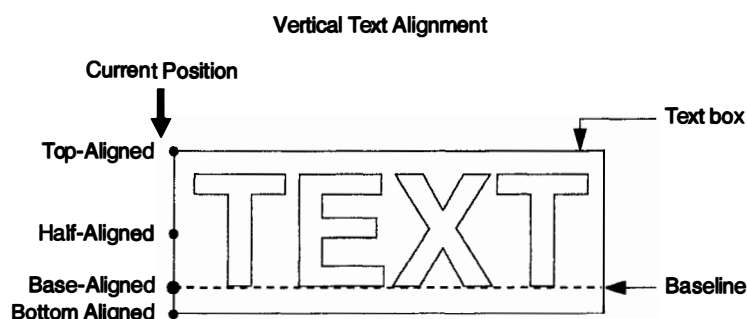


Figure 6-15. Vertical Positioning of Text Strings

The following flags are used to specify types of vertical alignment:

Table 6-4. Vertical Alignment Values	
Identifier	Alignment
TA_TOP	On the top edge of the topmost character in the string
TA_HALF	On the arithmetic mean of the topmost and bottommost characters in the string
TA_BASE	On the baseline of the bottommost character in the string
TA_BOTTOM	On the bottom edge of the bottommost character in the string

The *Presentation Manager Programming Reference* also describes 2 sets of default values for the **IVertical** option. They are provided for compatibility and map into the vertical alignment values described above.

Character Extra and Break Extra

Certain output devices permit you to specify extra space between character cells by using the *character extra* attribute. Sometimes the space between words also can be expanded by increasing the size of the *break character*, usually defined as the space character, by using the *break extra* attribute. If this adjustment to either attribute is permitted, the result will be in addition to the sizing effects caused by the following parameters:

- Font kerning
- Font proportional spacing
- Width vectors.

The break values you can specify create different effects, as follows:

Table 6-5. Break Values and Their Effects	
Value	Effect
Positive	Forces characters apart.
0	Resumes the default spacing created by other parameters.
Negative	Forces characters together, even overlapped characters.

These effects are illustrated in Figure 6-16.

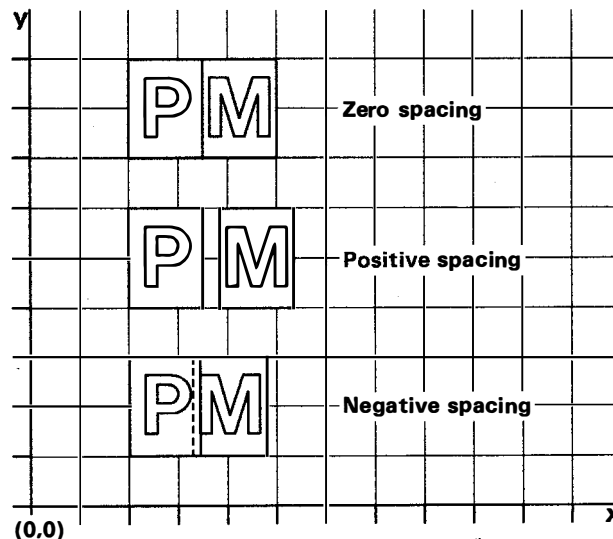


Figure 6-16. The Cumulative Effect of Break Values

The above are fixed integer values specified in world coordinates. Both the *character extra* and *break extra* attributes have initial default values of 0, which can be changed using `GpiSetDefAttrs` (`CBB_EXTRA`) and (`CBB_BREAK_EXTRA`). These values can be changed using `GpiSetCharExtra` or `GpiSetCharBreakExtra`, respectively; and the values can be queried using `GpiQueryCharExtra` and `GpiQueryCharBreakExtra`.

Character Color and Mix

The *color* attribute defines the color used to draw a primitive or an object. The *mix* attribute determines how the color of a primitive or an object is combined with the color of the drawing surface or any other objects on the surface. Both of these attributes are described in detail in Chapter 7, "Color and Mix Attributes."

The character-string color defines the color used to draw the output from any of the draw-character-string functions. When a presentation space is created, the character-string color default is black. Character strings are one of the primitives that have both a foreground and a background color, as shown in Figure 6-17.

For image characters, colors are determined by setting pels. For outline characters, the foreground consists of arcs and lines that define the character; the background color appears between the foreground lines. The character can be solid or filled, in which cases the background color does not appear between the foreground lines.

Character string primitives have a color attribute for both the actual character and its character cell, which surrounds the character. The character-cell color is the background color.

The foreground mix attribute controls the combination of character-string color and drawing-surface color, while the background mix attribute controls the combination of the character-cell color and the drawing-surface color, as illustrated in Figure 6-17.

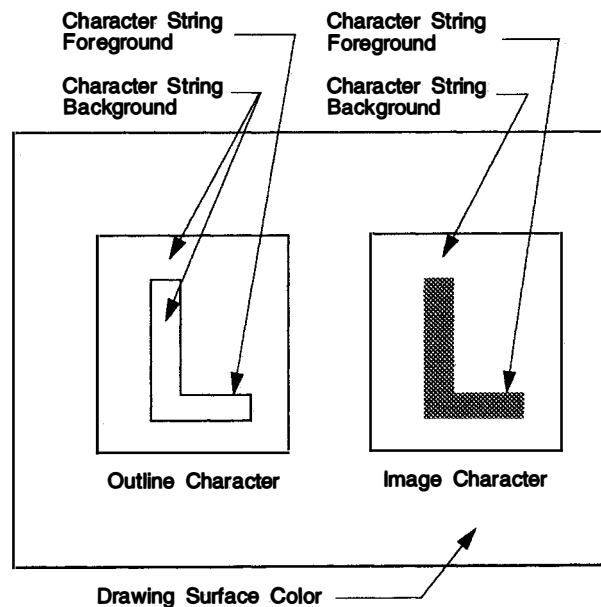


Figure 6-17. Character String Primitives

When a presentation space is created, the character string mix attribute default is FM_OVERPAINT. The *overpaint* mix attribute specifies that the character-string color is not to be modified by the color of the drawing surface. If the character string mix attribute is changed, the character-string color is mixed with colors that are already on the drawing surface.

The character string background color default is CLR_BACKGROUND, usually defined by the application as the same color as the drawing surface. The character string background mix attribute default is BM_LEAVEALONE. The *leave-alone*

background mix attribute specifies that the character string background color not be drawn. The cell that surrounds the character string appears only if the background character-string color and mix attributes are changed.

Use `GpiSetAttrs` to specify a new color or mix attribute. As input, this function accepts the following:

- Type of primitive, for example, `PRIM_CHAR`
- List of attributes to be changed
- List of attributes to be set to their default values
- Values for the attributes to be changed.

`GpiSetAttrs` also is useful to specify color and mix attributes for a specific data structure, for example, `CHARBUNDLE`. `GpiSetAttrs` provides some protection against invalid colors, as described in the *Presentation Manager Programming Reference*.

To determine the current character-string color and mix attributes, call `GpiQueryAttrs`, which accepts as input the primitive type and the attributes in question. `GpiQueryAttrs` returns an array of values for the queried attributes.

To reset the default character-string color and mix attributes, as with all attributes specified in `CHARBUNDLE`, call `GpiSetDefAttrs`, which accepts as input the type of primitive, attributes to be changed, and values that will become the new default values. Changing default values is especially important when working with segments, described in Chapter 12, “Creating and Drawing Retained Graphics” and Chapter 13, “Editing Retained Graphics and Graphics Segments.” Changing the default values during a series of drawing functions is not recommended.

The character color and mix attributes also can be specified using the following functions:

- `GpiSetColor`
- `GpiSetMix`
- `GpiSetBackColor`
- `GpiSetBackMix`.

If the character color, character background color, mix, or background mix attributes are specified individually, the following queries can return a value inconsistent with the current character attributes:

- `GpiQueryColor`
- `GpiQueryMix`
- `GpiQueryBackColor`
- `GpiQueryBackMix`.

Using Character String Primitives

You can use the character string primitive functions to perform the following tasks:

- Draw a string of characters using the selected font.
- Modify the string by using one or more of the following operations:
 - Scaling
 - Shearing
 - Rotating
 - Transforming
 - Changing the angle of the baseline.
- Aligning Text.

Drawing Text

Figure 6-18 shows how to select a Helvetica outline font, set the size of the character box, change the foreground color to red, set the character angle, and move the cursor to a specified location. Then, `GpiCharString` is used to write a string of characters with the specified size, color, angle, and location.

```
#define INCL_GPIPRIMITIVES
#define INCL_GPILCIDS
#include <os2.h>
void fncFONT09(void){

    POINTL ptl = { 100, 50 };
    GRADIENTL grad = { 4, 1 };
    SIZEF sizfx;
    FATTRS fat;
    CHARBUNDLE cbnd;
    FONTMETRICS afm[80];
    HPS hps;
    HDC hdc;
    LONG cHelvFonts, i;
    LONG cFonts = 0;
    LONG lcid = 1;
    LONG devRes[2];                /* Horizontal, vertical font resolutions */

    cHelvFonts = GpiQueryFonts(hps, QF_PUBLIC, "Helv",
                              &cFonts, sizeof(FONTMETRICS), NULL);

    GpiQueryFonts(hps, QF_PUBLIC, "Helv", &cHelvFonts,
                  sizeof(FONTMETRICS), afm);

                                /* Find an outline Helvetica font. */

    for (i = 0; (!(afm[i].fsDefn & FM_DEFN_OUTLINE)) &&
          i < cHelvFonts; i++) ;
```

Figure 6-18 (Part 1 of 2). Drawing Text

```

fat.usRecordLength = sizeof(FATRS);
fat.fsSelection = 0;
fat.lMatch = afm[i].lMatch;
StringCopy(fat.szFacename, afm[i].szFacename);
fat.idRegistry = 0;
fat.usCodePage = 0;
fat.lMaxBaselineExt = 0;
fat.lAveCharWidth = 0;
fat.fsType = 0;
fat.fsFontUse = FATR_FONTUSE_OUTLINE;

GpiCreateLogFont(hps, (PSTR8) NULL, lcid, &fat);
GpiSetCharSet(hps, lcid);

DevQueryCaps(hdc, CAPS_HORIZONTAL_FONT_RES, 2L, devRes);
sizfx.cx = MAKEFIXED((afm[i].sNominalPointSize * devRes[0]) / 720, 0);
sizfx.cy = MAKEFIXED((afm[i].sNominalPointSize * devRes[1]) / 720, 0);
GpiSetCharBox(hps, &sizfx);

cbnd.lColor = CLR_RED;
GpiSetAttrs(hps, PRIM_CHAR, CBB_COLOR, NULLHANDLE, &cbnd);

GpiSetCharAngle(hps, &grad);
GpiMove(hps, &pt1);
GpiCharString(hps, 11, "Vector Text");
} /* fncFONT09 */

```

Figure 6-18 (Part 2 of 2). Drawing Text. Certain parameters in the above example are explained in Chapter 9, "Fonts."

Formatting Text

Graphics text, like all other graphics objects, has to be positioned correctly in the output area, which usually consists of one of the following:

- Entire client area of a PM window
- Part of a PM window
- Addressable area of a printer.

Unlike other graphics objects, however, text is governed by well-established readability and usability rules. These rules apply generally to text output, whatever its method of production. Following are some recommendations:

- Lines of text from fonts with large point sizes must be more widely spaced for maximum readability.
- The longer the line of text, the greater the space between lines must be to ensure that the lines do not appear to merge.
- Very small text must be split into multiple columns rather than continued across the page.

Some of these considerations are taken care of by the font designer. The PM programming interface enables you to control both the horizontal and vertical positioning of text.

Positioning Text in World Coordinates

When considering text alignment, take the versatility of the coordinate systems into account. The following definitions depend on the current coordinate system:

Table 6-6. World Coordinate Values	
Value	Corresponds to the direction of...
Left	The lowest x-coordinate value
Right	The highest x-coordinate value
Top	The highest y-coordinate value
Bottom	The lowest y-coordinate value

To position a character string horizontally, you must know the width of the output area and the length of the character string. The PM programming interface provides 3 different functions for determining the width of the output:

Table 6-7. Functions for Determining Width of Character String Primitives	
Function Name	Description
GpiQueryGraphicsField	Returns the bottom-left and top-right corners of the graphics field in presentation page units.
GpiQueryViewingLimits	Returns the viewing limit.
GpiQueryPageViewport	Returns the page viewport in device units.

All three functions are described in the *Presentation Manager Programming Reference*.

The GpiConvert function changes coordinates into world coordinates. To calculate the width of the output area, subtract its **left** from its **right**. For example, if the **left** is 30, and **right** is 600, the width of the output area is 570 world coordinates.

The PM programming interface provides three different functions for determining the length of the character string primitive:

- GpiQueryTextBox
- GpiQueryCharStringPos
- GpiQueryCharStringPosAt.

GpiQueryTextBox returns the relative coordinates of a parallelogram that surrounds the character string. By subtracting the x-coordinate of TXTBOX_BOTTOMRIGHT from the x-coordinate of TXTBOX_BOTTOMLEFT, an application can determine the length of the string.

GpiQueryCharStringPos returns an array of points, in which the world coordinate position of each character in the string is recorded. The last value in the array becomes the new current position if the string is drawn using GpiCharStringPos. By subtracting this position from the current position (obtained using GpiQueryCurrentPosition) the length of the string can be determined.

GpiQueryCharStringPosAt also returns an array of points, in which the world coordinate position of each character in the string is recorded. The last value of the array becomes the new current position if the string is drawn using GpiCharStringPosAt. This function accepts a specified starting position for the

character string. By specifying a starting position of (0,0) for example, an application can determine the length of the string without subtraction.

The current position actually is not updated by either `GpiQueryCharStringPos` or `GpiQueryCharStringPosAt`. All three functions are described also in the *Presentation Manager Programming Reference*.

When a character string does not fill the width of the output area, it can be positioned in 1 of the 4 ways illustrated in Figure 6-19.

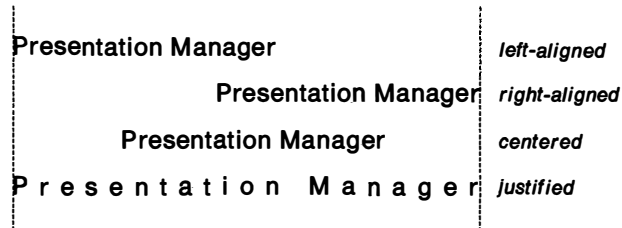


Figure 6-19. Horizontal Positioning of Text Strings

To left-align the text, set the x-coordinate of the current position to the *left* of the output area before drawing the character string. `GpiSetCurrentPosition` must be used to set the current position if your application draws the string using either `GpiCharString` or `GpiCharStringPos`. Both `GpiCharStringAt` and `GpiCharStringPosAt` accept a starting position as input.

To right-align the text, subtract the length of the character string from the width of the output area, then add the difference to the x-coordinate of the current position before drawing the character string. If the output area is 570 world coordinates wide, for example, and the character string is 436 world coordinates long, add 134 to the x-coordinate of the current position before drawing the text.

To center the text, subtract the length of the character string from the width of the output area, then divide the difference by 2 before adding it to the x-coordinate of the current position. If the difference is 134, for example, you add 67 to the x-coordinate of the current position.

To justify the text, so that the text string fills the width of the output area, distribute the surplus space throughout the character string. For example, you could add the extra space to the break characters only, or you could share it equally among all characters in the string. Text justification requires the individual positioning each character in the string using either of the following calls:

- `GpiCharStringPos`—draws at the current position and permits you to position every character after the first.
- `GpiCharStringPosAt`—permits you to position every character, including the first.

Both functions enable you to specify a different character increment for each character in the string. Distance is measured from the character reference point of one character to the character reference point of the next character. The values you specify apply only to the character string supplied at input; they *do not* become current attribute values.

If you are formatting a block of text, the string might be wider than the output area, or longer than 256 characters. In either case, your application must separate the string into smaller groups of characters so that it fits either criteria. A good starting point is to determine the number of characters planned for each line. Dividing the

output width by the character cell width can provide a first estimate as to where to separate a character string. An application can use this estimate to work through the string looking for spaces. Each time you find a space, compare the length of the string (up to the space) with the width of the output area. When the string is longer than the output area, work back to the previous space and display or print that part of the string. You can format the entire block of text for the output area in this way.

When you are formatting a block instead of a single line of text, an application must specify the vertical placement of each line. If you are using an image font, you have the assurance that each character is the same height. However, you do not have that assurance with an outline font, nor that the text block is written in the same font. Therefore, when calculating the separation of lines, avoid using complex combinations of font metrics values. Instead, it is recommended that you multiply the desired point size, or em-height metric, of the text by 1.2.

Summary

Table 6-8 summarizes the character string primitive functions.

<i>Table 6-8 (Page 1 of 2). Character String Primitive Functions</i>	
Function Name	Description
GpiCharString	Draws a character string using the current font from the current position.
GpiCharStringAt	Draws a character string using the current font from a specified position.
GpiCharStringPos	Draws a character string using the current font with the first character at the current position, then allows positioning of every following character.
GpiCharStringPosAt	Draws a character string in the current font and allows positioning for the first and every following character.
GpiQueryCharAngle	Determines the current character angle attribute.
GpiQueryCharBox	Determines the current character box attribute.
GpiQueryCharBreakExtra	Determines the current character break extra attribute.
GpiQueryCharDirection	Determines the current character direction attribute.
GpiQueryCharExtra	Determines the current character extra attribute.
GpiQueryCharMode	Determines the current mode attribute.
GpiQueryCharSet	Determines the current character set attribute.
GpiQueryCharShear	Determines the current character shear attribute.
GpiQueryCharStringPos	Determines the position of each character for an equivalent GpiCharStringPos function without actually drawing any characters.
GpiQueryCharStringPosAt	Determines the position at which each character will appear if an equivalent GpiCharStringPosAt function is used.
GpiQueryDefCharBox	Determines the size of the default character box.

<i>Table 6-8 (Page 2 of 2). Character String Primitive Functions</i>	
Function Name	Description
GpiQueryTextBox	Determines the coordinates of the corners of a text box for an equivalent GpiCharStringPos function without actually drawing any characters.
GpiSetCharAngle	Sets the current character angle attribute.
GpiSetCharBox	Sets the current character box attribute.
GpiSetCharBreakExtra	Sets the current character break extra attribute.
GpiSetCharDirection	Sets the current character direction attribute.
GpiSetCharExtra	Sets the current character extra attribute.
GpiSetCharMode	Sets the current character mode attribute.
GpiSetCharSet	Sets the current character set attribute.
GpiSetCharShear	Sets the current character shear attribute.

Table 6-9 summarizes the data structures used by the character string primitive functions.

<i>Table 6-9. Character String Primitive Structures</i>	
Structure Name	Description
CHARBUNDLE	Contains the character attributes.
FATTRS	Contains the attributes of the font specified using GpiCreateLogFont.
FONTMETRICS	Contains all the font metrics returned by the system to describe a particular font.
POINTL	Specifies the values of a single pair of x- and y-coordinates.

Chapter 7. Color and Mix Attributes

This chapter describes color and mix attributes and their use in OS/2 applications. The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Line and arc primitives
- Marker primitives
- Area primitives
- Character string primitives
- Bit maps and images primitives.

About Color and Mix Attributes

Color and mix are 2 attributes of graphics primitives. They can be specified by an application in a number of ways; and are specified in the `xxxBUNDLE` data structure associated with the 5 graphics primitives and in bit maps. Bit maps and some of the primitives have both foreground and background color attributes. For example, a character string primitive has a foreground color attribute that specifies the color of the character and a background color attribute that specifies the color of the character cell surrounding the character.

The mix attribute controls how each primitive is combined with the existing drawing. Among other things, it affects the color that results when primitives of different colors overlap. Primitives with foreground and background color attributes also have foreground and background mix attributes.

Color Implementation

Understanding how colors are implemented can assist in understanding how the color and mix attributes work.

The *pel* is the smallest element of the display screen that can be addressed. For monochrome displays, pels are either turned off or turned on. For color displays, each pel contains a red, green, and blue section, each of which is called a *phosphor*.

The display has color guns of red, blue, and green light that illuminate the phosphors in a single pel. By switching these color guns on and off in different combinations, 8 standard colors can be produced. By varying the intensities of the color guns, additional colors can be produced. Table 7-1 on page 7-2 shows the 8 standard colors that can be generated from the 3 color guns.

Table 7-1. Eight Standard Colors			
Pel appears...	Red	Green	Blue
Red	ON	OFF	OFF
Green	OFF	ON	OFF
Blue	OFF	OFF	ON
White	ON	ON	ON
Black	OFF	OFF	OFF
Cyan (Turquoise)	OFF	ON	ON
Pink	ON	OFF	ON
Yellow	ON	ON	OFF

Each pel is described internally by a number of bits of storage. In a monochrome display, only 1 bit per pel is required, and that bit is either on or off. In an 8-color system, 3 bits per pel are necessary. Each of those 3 bits records the on or off setting of 1 color gun.

To be able to control the intensity of a color and obtain more than 8 colors, more than 3 bits per pel are needed. For example, 6 bits per pel provide 32 different combinations.

The wider the range of available colors, the more bits per pel required for each color. This storage issue is resolved by keeping a wide choice of colors but restricting the number of colors available at any given time. Applications define the colors that they want to use in a *logical color table*. Selecting a color defined in the logical color table produces the nearest available color in the hardware palette.

RGB Color Encoding

The red, green, and blue (RGB) components of a color are stored in either an RGB or RGB2 data structure, or as a long integer (32-bit) value. The color fields in the RGB2 structure and in the long integer follow the same rules.

If stored as a long integer, the RGB value has the first 8 bits reserved for a flag value and the remaining 24 bits reserved for color intensity. The flag byte must be set to 0. Each of the last 3 bytes specifies a color intensity, in the range 0 through 255, for a single primary color.

If a byte contains 0, the corresponding primary color is not present. As the value in the byte increases, the intensity of the primary color increases. For example, if the byte contains 128, the primary color is pale; if the byte contains 255, the primary color is as intense as the device permits.

The RGB value is determined by the following equation:

$$\text{RGB Value} = (R * 65536) + (G * 256) + B$$

Where:

R is the red intensity
G is the green intensity
B is the blue intensity

If all 3 bytes are set to 0, the resulting color is black; if they all are set to 255, the resulting color is white. The RGB value associated with each of the standard 8 colors is in Table 7-2 on page 7-3.

Table 7-2. RGB Values	
Color	RGB value
Black	0x00000000
Red	0x00FF0000
Green	0x0000FF00
Blue	0x000000FF
Pink	0x00FF00FF
Cyan	0x0000FFFF
Yellow	0x00FFFF00
White	0x00FFFFFF

Color Tables

A color table is an array of RGB values. There are 2 kinds of color tables: logical and physical. The logical color table is a list of colors specific to a presentation space. An application typically uses a logical color table to define colors specific to that application.

The physical color table specifies colors the device can generate currently. These device colors are shared by every application on the system. Because some display adapters cannot generate every possible device color at the same time, the physical color table can be a subset of the full range of possible device colors. The operating system maps the RGB values specified in the logical color table to device colors in the physical color table.

Logical Color Table

A logical color table contains a variable number of entries, each of which describes a different RGB (Red, Green, Blue) combination that produces a color. The principle of the color table is illustrated in Figure 7-1.

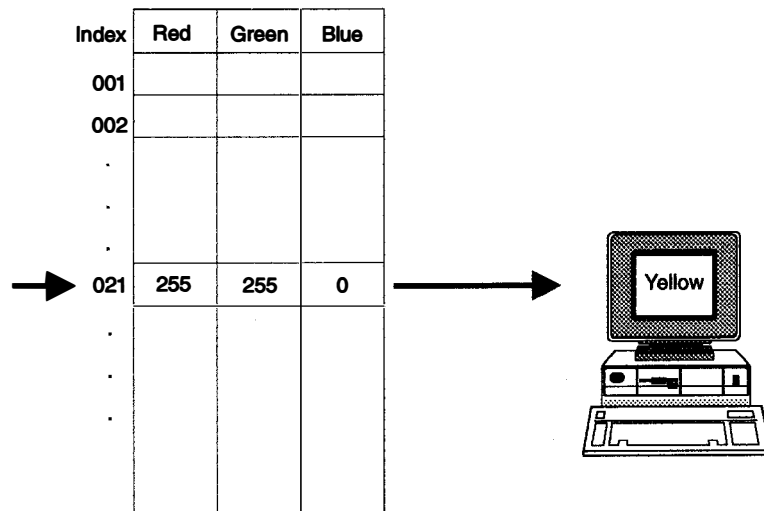


Figure 7-1. Logical Color Table. This simplified example demonstrates that to produce yellow on the computer screen, red and green are mixed in equal intensities, and no blue is used at all. In this example, yellow is addressed in the color table by index 21. Notice that this is not the same index number used in the default logical color table, which indicates that this color table has been changed by the application.

A logical color table is stored in a presentation space and is specific to that presentation space. A logical color table enables applications to specify colors as indexes rather than explicit RGB values.

The colors displayed are likely to vary from 1 output device to another so the definitions in the color table can be fine-tuned to get the best results. Any color can be made more or less intense by altering its definition in the logical color table.

Default Logical Color Table

The PM programming interface provides a default logical color table, which defines the colors and the indexes that retrieve them, as shown in Table 7-3.

<i>Table 7-3. Default Logical Color Table</i>		
Color Index	Index Number	Effect
CLR_FALSE	-5	All bits are set to 0.
CLR_TRUE	-4	All bits are set to 1.
CLR_DEFAULT	-3	Default value
CLR_WHITE	-2	White
CLR_BLACK	-1	Black
CLR_BACKGROUND	0	Natural background color for the device
CLR_BLUE	1	Blue
CLR_RED	2	Red
CLR_PINK	3	Pink
CLR_GREEN	4	Green
CLR_CYAN	5	Cyan
CLR_YELLOW	6	Yellow
CLR_NEUTRAL	7	Neutral — The contrasting color to CLR_BACKGROUND
CLR_DARKGRAY	8	Dark gray
CLR_DARKBLUE	9	Dark blue
CLR_DARKRED	10	Dark red
CLR_DARKPINK	11	Dark pink
CLR_DARKGREEN	12	Dark green
CLR_DARKCYAN	13	Dark cyan
CLR_BROWN	14	Brown
CLR_PALEGRAY	15	Pale gray
Note: Entries after CLR_PALEGRAY have device-dependent defaults.		

PM maps the color index name to the index number (shown in the second column of the previous table), then uses that number to address the appropriate color.

Device-Independent Color Indexing

Three of the index names provide a level of device independence in choosing colors: CLR_DEFAULT, CLR_BACKGROUND, and CLR_NEUTRAL. These indexes enable an application to select colors according to their purpose, and thus build device independence into your applications. The purpose of a color does not vary from 1 device to another, although the actual color used to implement that purpose might. The following table describes these indexes and the purpose of each:

Table 7-4. Device Independent Color Indexes	
Index	Purpose
CLR_BACKGROUND	The natural background color for the device. This is the color of the paper on a printer, and the window background color (white, by default) on a display.
CLR_NEUTRAL	The contrast color to CLR_BACKGROUND. This is usually black on a printer, and the default window text color (black, by default) on a display.
CLR_DEFAULT	Unless redefined, this has the same effect as CLR_NEUTRAL.

The colors produced by CLR_DEFAULT, CLR_BACKGROUND, and CLR_NEUTRAL in the default logical color table depend on the output device. For example, CLR_NEUTRAL could produce black on a device with a white background or white on a device with a black background.

Defining a Logical Color Table

To change the values in the default logical color table (that is, to load a new logical color table), use GpiCreateLogColorTable. Using this function, an application can do the following:

- Replace part or all of the default color table.
- Add color definitions to the default color table.
- Reset the logical color table to its default values.

There are 2 methods of making changes:

1. To add to a table or change some of its contents, rather than replace the table completely, supply an array of color indexes and their associated RGB values.
2. To load a consecutive sequence of index values, supply only an array of RGB values without index values.

Color Tables in Index Mode

The default logical color table is defined as an array of color indexes and their associated RGB values. A table in this format is in *index mode*.

To alter a logical color table in index mode, specify either `LCOLF_INDRGB` or `LCOLF_CONSECRGB` as the format value in `GpiCreateLogColorTable`. If an application uses the color index names to address the contents of a loaded color table in index mode, the color identified by the associated index number is retrieved. For example, `CLR_NEUTRAL` produces the color addressed by index number 7. If `CLR_NEUTRAL` must be blue, the application must define the color table so that index number 7 addresses the RGB definition of the appropriate shade of blue.

If an application uses the format `LCOLF_INDRGB` when calling `GpiCreateLogColorTable`, the application supplies an array of index and RGB pairs to update the table. These values do not have to be consecutive. An application can change the contents of an existing table and add entries to the end of the table.

If an application calls `GpiCreateLogColorTable` using the `LCOLF_CONSECRGB` format, the application supplies an array of RGB values and a starting index. When using this format, the RGB values must be consecutive; you cannot use a single call to `GpiCreateLogColorTable` to redefine `CLR_BLUE` and `CLR_PINK` without also specifying an RGB value for `CLR_RED`. As with `LCOLF_INDRGB`, this format can be used to change the contents of the existing table and to add entries to the end of the table.

When an application calls `GpiSetAttrs` to put a value into the color field of a graphics primitive attribute structure, the value is an index into the logical color table. When the operating system draws the primitive, it uses this index to determine the RGB value specified in the logical color table by the application; then it searches the physical color table for the color closest to this RGB value. The operating system then draws the primitive, using the color from the physical color table. How closely the drawn colors match the colors in the logical color table depends on the device colors in the physical color table.

Color Tables in RGB Mode

The color values specified with `GpiSetAttrs` and `GpiSetColor` also can be specified directly as RGB values rather than as indexes. To enable this, the logical color table must be switched into RGB mode by calling `GpiCreateLogColorTable`, specifying a format of `LCOLF_RGB`. (No color array is passed.)

If the color table is in RGB mode, it can be switched back to index mode and reset to its original default values by specifying either `LCOLF_INDRGB` or `LCOLF_CONSECRGB` in `GpiCreateLogColorTable`.

Querying the Available Colors

Several query functions are provided for applications to get information about the current logical and physical color tables before defining a logical color table.

Applications use `GpiQueryColorData` to determine whether the default logical color table is in effect. If it is not, the following information about the loaded logical color table is returned:

- Format of the current logical color table
- Smallest and largest color indexes (if the table is in index mode) supported in this color table. The smallest color index is always 0, and the largest color index is never less than 15, because deleting entries from the default logical color table is not permitted.

`GpiQueryRealColors` returns the RGB values of each of the distinct colors defined in the physical color table of the current device. It also can return the index in the current logical color table that references each of the colors in the physical table.

To determine the available colors nearest to a specified color on a particular device, call `GpiQueryNearestColor`, which accepts as input the RGB value of the desired color. The function returns the RGB value of the nearest available color in the physical color table of the associated device.

Applications can use `GpiQueryRGBColor` to determine the RGB value of a particular color. `GpiQueryRGBColor` accepts as input the index to the logical color table entry of the color in question. Output from this function is the RGB value that the index would reference in the physical color table. If a logical color table is loaded in RGB mode, `GpiQueryRGBColor` returns the same results as `GpiQueryNearestColor`.

Conversely, to determine the index value that references a given RGB color (or the closest match to that color) in the physical color table, an application uses `GpiQueryColorIndex`. The application can determine which colors are in the current logical color table by calling `GpiQueryLogColorTable`. To determine which colors are in the physical color table, call `GpiQueryRealColors`.

Physical Color Table

Each output device has its own *physical color table*, which is organized like a logical color table. The physical color table contains the RGB color definitions of the distinct colors that the device can produce, while a logical color table contains the definitions of the colors as chosen by an application. When an application draws on an output device, the PM programming interface maps the index value that addresses the current color in the logical color table to the index that retrieves the closest match for the current color in the physical color table.

Because this substitution occurs when the graphics are drawn, the presentation space can be associated with a number of different device contexts without invalidating the logical color definitions. For example, an application can create a picture on the screen using a wide range of colors, then direct that drawing to an 8-pen vector plotter. The drawing is reproduced without having to be re-created. Although the picture, as drawn by the plotter, does not have the variety of colors displayed on the screen, the substitution process selects the nearest match for each color on the screen from the 8 available colors on the plotter.

Palette Manager

Applications can use `GpiCreatePalette` to change the physical color palette by creating a new palette. This function is used if the application has specific color needs and must ensure that the color is part of the physical color palette. `GpiCreatePalette` also enables an application to prevent *dithering* of the 16 default colors by setting the flag parameter to `LCOL_PURECOLOR` (for information on dithering, see “Dithering” on page 7-17). If the full 256 colors in the palette are needed, the application can set the flag to `LCOL_OVERRIDE_DEFAULT_COLORS`. It is recommended that the Palette Manager functions be used only when really necessary because the operating system cannot guarantee consistent colors in other windows and the ability to change the physical palette is device dependent.

After the palette is created, an application can set or change any of the values in the palette by calling `GpiSetPaletteEntries`. To delete the palette, applications use `GpiDeletePalette`, passing the handle of the palette to be deleted. (The handle is returned by `GpiCreatePalette` when the palette is created.)

`GpiQueryPalette` enables applications to determine the palette currently selected for a given presentation space. A palette can be used by more than 1 presentation space at the same time. The operating system maintains a count of the number of presentation spaces using a specific palette. Complete information on the current palette can be accessed by `GpiQueryPaletteInfo`.

Palettes also can be shared by using a palette handle.

When coding an application, color table functions and color palette functions must not be mixed. The application can call `DevQueryCaps` to determine whether the hardware supports palettes, then call the appropriate functions.

Realizing a Color Palette

`WinRealizePalette` maps the colors requested by the application into the color palette for the system. When the application's window is activated, the palette changes are transferred, or *realized*, into the physical palette for the system.

When a palette is realized, there might not be enough empty palette entries to accommodate the additional palette changes (a maximum of 256 entries). For example, it is possible that some of the colors changed in the physical palette are being used by another application. In this situation, a `WM_REALIZEPALETTE` message is posted to all the applications running on the desktop.

On receiving the `WM_REALIZEPALETTE` message, applications using Palette Manager functions must repaint their screens. The original application colors are mapped to the closest matching color in the new palette. Applications that do not use Palette Manager functions normally perform default processing within their applications causing a repaint of their windows with the closest match from the palette. If there are no changes, just additions, to the physical palette, no message is sent.

Note: The Palette Manager maps a window's colors to the closest available value in the palette when the physical palette has to be changed, but this does not guarantee that the color will be a close match to the original color used.

As the focus changes from window to window, the physical palette changes according to the activated window. Notification messages are sent as necessary to other applications.

If the physical color values in the palette have to change to accommodate the number of palette entries passed from the application when a palette is realized, the number of altered entries in the physical palette is returned by `WinRealizePalette`.

Color palette realization is available only to systems that have a minimum of 256 colors.

Color Attribute

The PM graphic interface uses a variety of colors. These colors are referred to as the *system colors*, and they are defined in the system color table, which is separate from an application's logical color table. To find out the RGB values of the system colors, call `WinQuerySysColor`.

The colors of graphic primitives are specified separately from the system colors. Every primitive has a *foreground* color; some also have a *background* color.

Primitive Foreground

The foreground of a primitive is the primitive itself. For example, the foreground of a full arc primitive is the full arc, as shown in Figure 7-2.

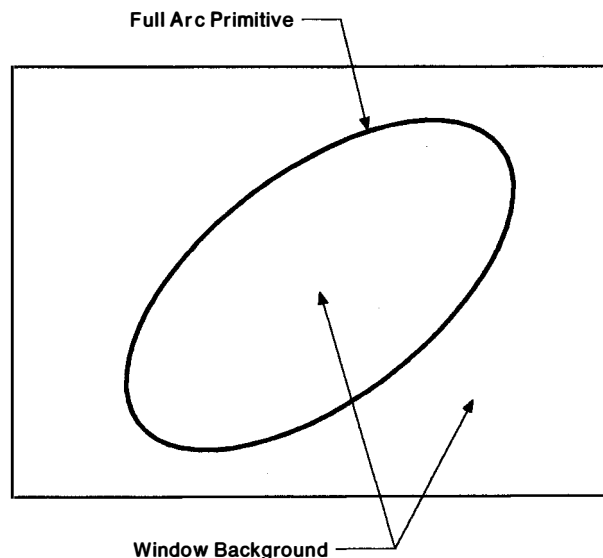


Figure 7-2. Foreground of a Primitive. The full arc primitive is drawn in a different color from the window background.

By default, the foreground color of all primitives is the color addressed by the index `CLR_DEFAULT`. In the default color table, this produces black on a graphics display. If the application replaces the default logical color table, `CLR_DEFAULT` produces the color addressed by index number 7 (`CLR_NEUTRAL`).

Primitive Background

The following primitives have a background:

- Areas
- Character strings
- Images
- Markers.

The background of any character or marker primitive, whether the primitive is from an image or an outline font, is the *entire* character or marker box. First the

background is drawn, then the foreground is drawn on top of it. Similarly, the background of an area primitive is the entire area to be filled. The background of an image primitive, however, is that part of the primitive in which the pels are not set. Figure 7-3 shows the background of a primitive.

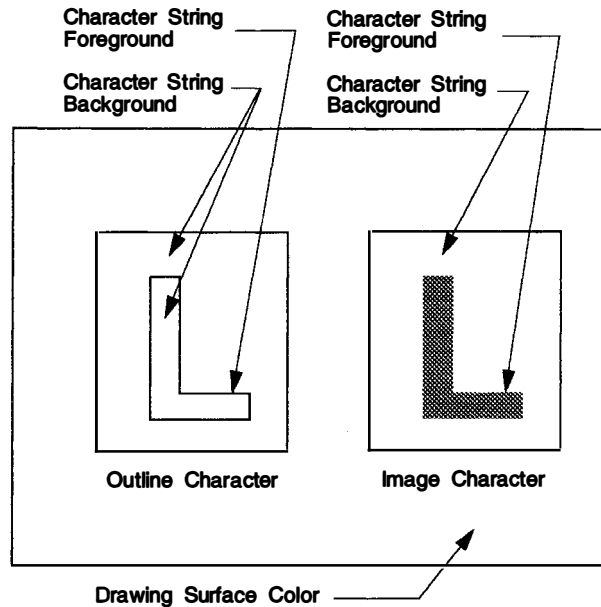


Figure 7-3. Background of a Primitive. The background of this character-string primitive is the entire character box.

The index to the default background color is `CLR_BACKGROUND`, which provides a background color appropriate for the device. On a printer, `CLR_BACKGROUND` is the color of the paper. On a display screen, `CLR_BACKGROUND` is the default window background color.

Changing the Foreground and Background Colors of Primitives

To change the current foreground color, use `GpiSetColor`; as input, the application supplies either the index to the required color in the current logical color table or the RGB value of the color, depending on the mode of the table. Color indexes higher than those supplied in the default color table must be loaded explicitly before they can be used.

It is possible to specify 1 of the system colors (for example, `SYSCLR_ACTIVEBORDER`) as the current foreground (or background) color of a primitive. The color appears as defined in the system color table, and the logical color table is not used.

The specified color becomes the current color, and the foreground of any primitive drawn subsequently is drawn in that color. The current foreground color for a particular primitive type is set by calling `GpiSetAttrs`. For example, if an application sets the current foreground color to `CLR_RED` by calling `GpiSetColor`, and sets the current foreground color for marker primitives to `CLR_CYAN` by calling `GpiSetAttrs`, all subsequent markers are cyan, and all other primitives are red.

To change the current primitive background color, call `GpiSetBackColor`. To ensure that this color is different from the background color of the output area (so that the entire primitive background is visible), the current window background color must

not be the same as the current primitive background color. Also, select an appropriate background mix attribute.

GpiQueryBackColor returns the current background color setting for a character primitive; to learn the current foreground color setting, use GpiQueryColor. GpiQueryAttrs can be used to determine the current foreground and background color values for a single primitive type.

Color Output and Mix Attributes

A *mix attribute* is a bitwise operation on the color indexes in a device's logical color table. Mix attributes enable colors to be specified in relation to other colors. When an application draws a graphics primitive, the operating system uses a mix attribute to determine the color that appears on the output device.

For example, instead of specifying that a line should be black and, therefore, invisible if the background also is black, an application can use a mix attribute of FM_XOR to specify that the line always should be drawn the inverse of the background color.

Suppose an application has set the **IColor** field in the LINEBUNDLE structure to CLR_RED and the **usMixMode** field in the same structure to FM_OR. The current color of the drawing surface is CLR_GREEN. To determine the color of a line, the operating system performs a bitwise OR operation on the indexes of these colors, as follows:

```
0010 (the default index for red)
0100 (the default index for green)
-----
0110 (result of bitwise OR)
```

In this case, the result is 6—the index for yellow. This means that even though the application specified CLR_RED in the LINEBUNDLE structure, a yellow line appears when the application calls any of the functions that draw a line primitive.

Mix Attribute

The mix attribute determines how each primitive an application draws is combined with any existing drawing. In color applications, the mix attribute determines the color that results when 1 primitive is drawn on top of another. There are 2 forms of the mix attribute: *foreground mix* and *background mix*.

The foreground mix attribute governs how the foreground of a primitive is combined with the existing drawing, and it applies to all primitive types. The background mix attribute governs how the background of a primitive is combined with the existing drawing, and it is applicable only to those primitives that have a background. Primitives that can be affected by the background mix attribute are areas, character strings, images, and markers. The primitive attribute data structures contain fields for both foreground and background color and mix attributes.

There are 16 foreground mix attributes. For each mix attribute, the indexes of the foreground and current drawing-surface colors are combined by using 1 of the bitwise operators. The available foreground mix settings are listed in Table 7-5 on page 7-12.

Table 7-5 (Page 1 of 2). Foreground Mix Attributes

Mix Attribute	Effect	Description
FM_DEFAULT	Default	Default foreground mix attribute (overpaint).
FM_OR	OR	Index value of the final color is determined by a bitwise OR operation on the index of the foreground color and the index of the color of the drawing surface.
FM_OVERPAINT	Overpaint	Index value of the final color is that of the foreground color. This is the default foreground mix attribute.
FM_XOR	Exclusive-OR (XOR)	Index value of the final color is determined by a bitwise XOR operation on the index of the foreground color and the index of the color of the drawing surface.
FM_LEAVEALONE	Leave-alone (Invisible)	Index value of the final color is that of the index of the color of the drawing surface.
FM_AND	AND	Index value of the final color is determined by a bitwise AND operation on the index of the foreground color and the index of the color of the drawing surface.
FM_SUBTRACT	(Inverse Source) AND Destination	Index value of the final color is determined by inverting the index of the foreground color and performing a bitwise AND operation on this value and the index of the color of the drawing surface.
FM_MASKSRCNOT	Source AND (Inverse Destination)	Index value of the final color is determined by inverting the index value of the drawing-surface color and performing a bitwise AND operation on this value and the index value of the foreground color.
FM_ZERO	All zeros	RGB value of the final color's is always 0x00000000.
FM_NOTMERGESRC	Inverse (Source OR Destination)	Index value of the final color is always the inverse of the FM_OR result.
FM_NOTXORSRC	Inverse (Source XOR Destination)	Index value of the final color is always the inverse of the FM_XOR result.
FM_INVERT	Inverse (Destination)	Index value of the final color is always the inverse of the index of the color of the drawing surface.
FM_MERGESRCNOT	Source OR (Inverse Destination)	Index value of the final color is determined by performing a bitwise OR operation on the index of the foreground color and the inverse of the index of the color of the drawing surface.

Table 7-5 (Page 2 of 2). Foreground Mix Attributes

Mix Attribute	Effect	Description
FM_NOTCOPYSRC	Inverse (Source)	Index value of the final color is the inverse of the index of the foreground color.
FM_MERGENOTSRC	(Inverse Source) OR Destination	Index value of the final color is determined by performing a bitwise AND operation on the index of the drawing surface's color and the inverse of the index of the foreground color.
FM_NOTMASKSRC	Inverse (Source AND Destination)	Index value of the final color is the inverse of the FM_AND result.
FM_ONE	All 1's.	RGB value of the final color is always 0x00FFFFFF.

There are 4 background mix attributes. For each mix attribute, the index value for the background color and the current drawing-surface color (in the device's physical color table) are combined using 1 of the bitwise operators.

The RGB values are those from the *physical* color table. When the result of the mix attribute's bitwise operation defines a color different from that of both the drawing surface and the drawing attribute, the resulting index accesses an RGB color in the physical table. The color, therefore, is unpredictable unless the logical color table has been realized (using the Palette Manager).

The first 5 of the foreground mix attributes also are available as background mix attributes. The background mix attributes are listed in Table 7-6.

Table 7-6. Background Mix Attributes

Mix Attribute	Effect	Description
BM_DEFAULT	Default	Default background mix attribute (Leave-alone).
BM_OR	OR	Index value of the final color is determined by a bitwise OR operation on the index of the background color and the index of the color of the drawing surface.
BM_OVERPAINT	Overpaint	Index value of the final color is that of the background color.
BM_XOR	Exclusive-OR (XOR)	Index value of the final color is determined by a bitwise XOR operation on the index of the background color and the index of the color of the drawing surface.
BM_LEAVEALONE	Leave-alone (Invisible)	Index value of the final color is that of the drawing-surface color.

The most frequently used foreground mix attributes are FM_OVERPAINT, which is the default value, FM_OR, and FM_XOR. The most frequently used background mix attributes are BM_LEAVEALONE, which is the default value, and BM_OVERPAINT.

Overpaint Mix Attribute

When using FM_OVERPAINT, the foreground of the primitive replaces any existing drawing in the same area of the presentation page. If the existing drawing is yellow, for example, and the new drawing is red, the drawing is red at the points of overlap. (This is the default foreground mix attribute.) Because 1 color is replacing another and no color mixing is being performed, the effects of the overpaint mix attribute are entirely predictable, as shown in Figure 7-4.

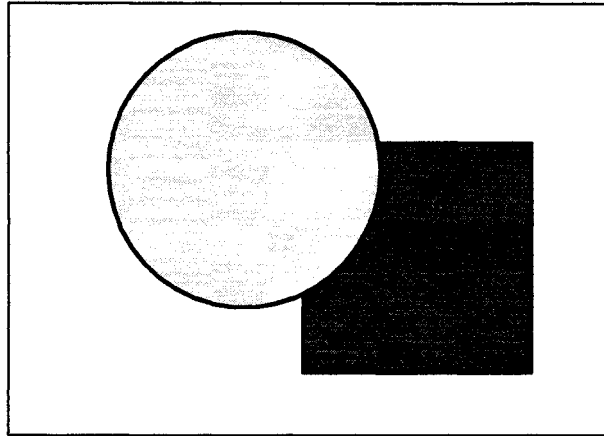


Figure 7-4. Overpaint Foreground Mix Attribute. The circle is drawn on top of the square. At the points of overlap, the output is the color of the circle.

When using BM_OVERPAINT, the primitive background replaces the existing drawing, as shown in Figure 7-5.

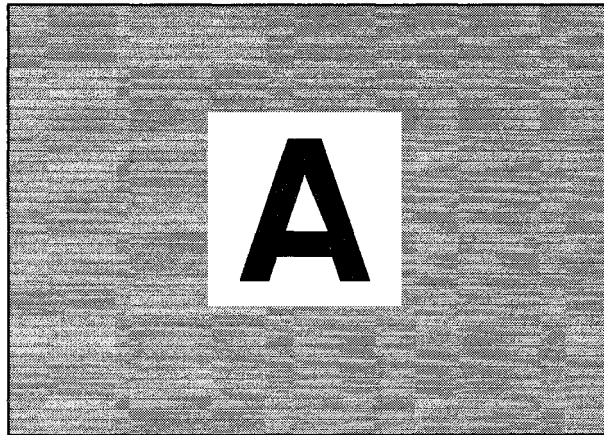


Figure 7-5. Overpaint Background Mix Attribute. Using BM_OVERPAINT, the background of the primitive is apparent only if it is drawn in a different color from the output-area background. Notice that, in this example, the foreground mix attribute is FM_OVERPAINT.

OR Mix Attribute

When using FM_OR, the foreground of the new primitive is merged with the existing drawing at the points of overlap. This is effected by ORing the indexes of the overlapping colors to produce a third color. The resulting color is unpredictable if the logical color table has not been realized (using the Palette Manager). The OR mix attribute is useful for making the common points of 2 graphics distinct from the points belonging to 1 of the graphics only, as shown in Figure 7-6 on page 7-15.

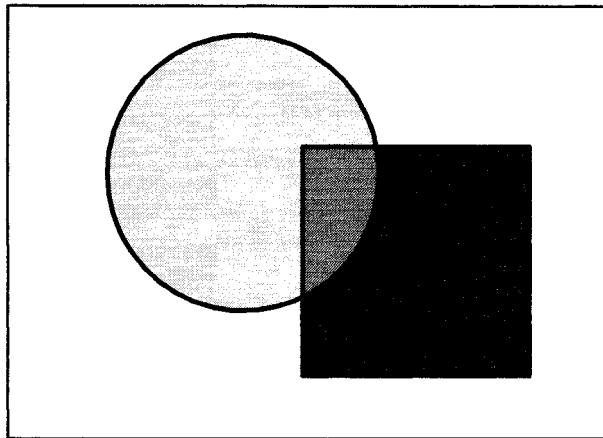


Figure 7-6. OR Mix Attribute. The circle is drawn on top of the square. At the points of overlap, indexes are OR'd to produce a new index referencing a new RGB color in the physical color table.

When using the BM_OR attribute, the background of the primitive is merged with the existing drawing according to the same rules that apply to the FM_OR attribute.

Exclusive-OR (XOR) Mix Attribute

The FM_XOR attribute enables objects to be drawn in such a way that they can be removed easily by simply drawing them a second time using the FM_XOR attribute. The FM_XOR attribute is available on display devices only and is useful for graphics animation when an application must move an individual graphic and completely restore the graphics that it originally overlapped. Typically, an application would do the following:

1. Draw the graphics object using the FM_XOR attribute.
2. Calculate the object's next position.
3. Draw the object again in its current position, still using the FM_XOR attribute.
This effectively erases it from its current position without destroying the graphics with which it overlaps.
4. Draw the object in its new position using the FM_XOR mix attribute.

For retained graphics, this sequence can be automated to some extent by defining specific graphics segments as *dynamic*. Dynamic graphics always are drawn using the FM_XOR attribute, regardless of the current mix attributes.

The effects of the FM_XOR attribute are shown in Figure 7-7.

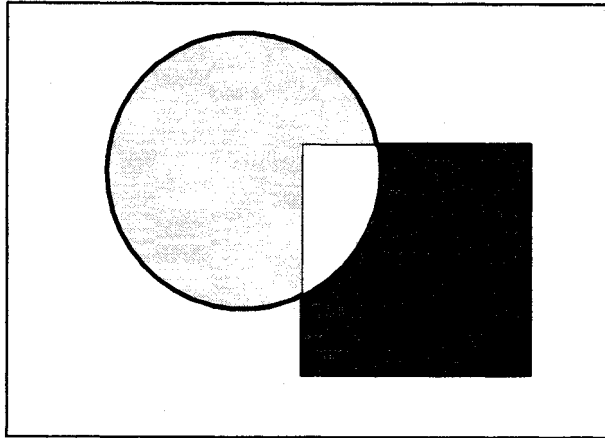


Figure 7-7. Exclusive-OR (XOR) Mix Attribute. The circle is drawn on top of the square. At the points of overlap, graphics are not drawn if the 2 overlapping figures are an identical color.

When using the BM_XOR attribute, the background of the primitive is merged with the existing picture according to the same rules that apply to the FM_XOR attribute.

Leave-Along Mix Attribute

The leave-alone mix attribute most often is used as a background mix attribute. When using the FM_LEAVEALONE attribute, the foreground of the primitive is not drawn. When using the BM_LEAVEALONE attribute, the background of the primitive is not drawn, as shown in Figure 7-8.

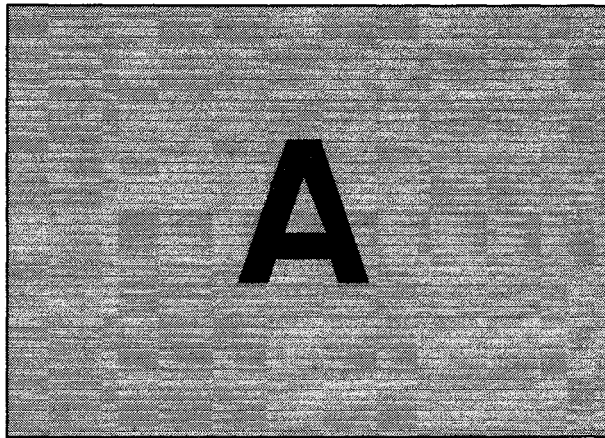


Figure 7-8. Leave-Along Background Mix Attribute. This is the background mix attribute that most often is used for character strings and marker primitives. It is generally used with a foreground mix value of FM_OVERPAINT.

Specifying Foreground and Background Mix Attributes

Specify the current foreground mix value for all primitives using GpiSetMix. To learn the current foreground mix setting, use GpiQueryMix, which returns the current foreground mix setting for a character string primitive. GpiQueryAttrs can be used to determine the current foreground and background mix values for a single primitive type.

To specify a background mix value for all primitives, use `GpiSetBackMix`. `GpiQueryBackMix` returns the current background mix value for a character-string primitive. To specify the current foreground and background mix values for a single primitive type, call `GpiSetAttrs`.

Note: Not all devices support all of the background and foreground mix attributes described. When a device does not support the mix attribute chosen by an application, the default mix attribute is used; no error condition is raised. `DevQueryCaps` can be used to determine whether a mix attribute is supported on a specific device.

Color on Advanced Display Devices

Some devices can display simultaneously a fixed number of colors (typically 156), chosen from a much larger number of colors (often more than 256000). An application can use the Palette Manager functions to take advantage of the extra capabilities of these devices. These functions enable an application to change the colors in a device's physical color table and the displayed colors rapidly without explicitly redrawing the screen.

An application can use the `CAPS_ADDITIONAL_GRAPHICS` option of `DevQueryCaps` to determine whether the display device supports palette functions.

Dithering

If an application requests a color not available in the physical color table, the operating system can approximate the color by a process called *dithering*. For example, if the physical color table does not contain a light green color but does contain a yellow and a green, the operating system can create what appears to be light green by mixing yellow pels and green pels. Dithering is a variation on the way red, green, and blue color guns illuminate the phosphors in a single pel to produce a color that is not pure red, green, or blue (for example, yellow).

The dithering process takes advantage of the way the human eye interprets color. If every other pel is set to 1 color, and all the intermediate pels to a different color, together they produce the effect of a third color at normal viewing distances.

The checkerboard effect is just 1 of the ways in which dithering can be implemented. Dithering works only when producing a solid mass of color, such as an area fill pattern. It does not have the desired effect on line primitives.

Dithering is especially important on monochrome devices. By combining various combinations of black pels with white pels, the operating system can create numerous shades of gray.

To use only the pure colors defined in the physical color table, that is, to prevent color dithering, set `LCOL_PURECOLOR` in `GpiCreateLogColorTable`. When `LCOL_PURECOLOR` is set, the nearest available color to the 1 selected is used.

Considerations When Using Monochrome Displays

When a graphic primitive that is drawn in color is displayed on a monochrome display, the operating system maps the colors that the application uses to the colors supported by the monochrome display.

Drawing Color Graphics on Monochrome Devices

When mapping color graphics to a monochrome device (screen, printer, or bit map), the monochrome device has a *reset* color and a *contrast* color. The reset color is the color GpiErase clears to and is either black or white. If the reset color is black, the contrast color is white; and if the reset color is white, the contrast color is black.

To determine whether the reset color is to be black or white, the color retrieved by the index CLR_BACKGROUND from the current logical color table is examined. If a logical color table has not been loaded, CLR_BACKGROUND is either the paper color on a printer or SYSCLR_WINDOW on a display. If a logical color table has been loaded, CLR_BACKGROUND is the color addressed by color index 0 on any device.

If the color retrieved by CLR_BACKGROUND is either white or black, that color becomes the reset color. For example, if an application is drawing to a screen, has not loaded a color table, and the system colors have not been altered, the background color is white, because SYSCLR_WINDOW produces a white background by default. In this instance, the contrast color is black.

If the color retrieved by CLR_BACKGROUND is neither white nor black, the color translates to whichever of black and white it is nearest to. As a rule, dark colors produce a black background, and pale colors produce a white background. For example, if an application is using a loaded color table, or if the color retrieved by SYSCLR_WINDOW has been altered (either interactively or by WinSetSysColors), CLR_BACKGROUND could be dark gray. In this instance, the reset color would be black, and the contrast color would be white.

When the reset color has been established, the PM programming interface applies the following rules when mapping color graphics to a monochrome device:

- Any color graphics drawn in CLR_BACKGROUND, and any graphics defined in the actual reset color (which is either black or white), are drawn in the reset color. Any other graphics are drawn in the contrast color.
- The index CLR_WHITE produces white, and the index CLR_BLACK produces black, regardless of whether the reset color is black or white.
- If no color table is loaded and CLR_DEFAULT or CLR_NEUTRAL are used as the foreground color, they produce the contrast color. If they are used as the background color, they produce the reset color.
- If an application calls GpiQueryNearestColor for a monochrome device, one of the following occurs:
 - If the color supplied on input is the reset color, the reset color is returned.
 - If the color supplied on input is not the reset color, the contrast color is returned.

Drawing Color Area Fill Patterns on Monochrome Devices

An area primitive is drawn according to the current foreground and background mix attributes and in the current area foreground and area background colors.

When an application draws a monochrome pattern on a color device, the bits of the pattern set to 1 translate to the current area foreground color, and the 0 bits translate to the current area background color. When the application draws a color pattern on a monochrome device, and if the current pattern is anything other than PATSYM_DEFAULT or PATSYM_SOLID from the default pattern set, the color closest to white is translated into 1 bits. For example, if a pattern of diagonal lines is being drawn in which the foreground color is red and the background color is cyan, the

cyan is translated to white (1 bits) because cyan is closer than red is to white. Red, therefore is translated to black (0 bits). The effect of translating this color pattern to a monochrome surface is summarized as follows:

Pattern	As 1s and 0s	Color Surface	Monochrome Surface
	10001000	RcccRccc	01110111
	01000100	cRcccRcc	10111011
	00100010	ccRcccRc	11011101
	00010001	cccRcccR	11101110
	10001000	RcccRccc	01110111
	01000100	cRcccRcc	10111011
	00100010	ccRcccRc	11011101
	00010001	cccRcccR	11101110

The original pattern of 1's and 0's is used, however, when deciding which part of the pattern is the background and which part is the foreground. Thus, if the background mix attribute is BM_LEAVEALONE, the following occurs:

Pattern	As 1s and 0s	Color Surface	Monochrome Surface
	10001000	R...R...	0...0...
	01000100	.R...R..	.0...0..
	00100010	..R...R.	..0...0.
	00010001	...R...R	...0...0
	10001000	R...R...	0...0...
	01000100	.R...R..	.0...0..
	00100010	..R...R.	..0...0.
	00010001	...R...R	...0...0

The 1 bits on the monochrome surface still are interpreted as the background of the primitive and are not drawn when the BM_LEAVEALONE attribute is specified.

When a bit map is used as an area fill pattern, any bit drawn in the current area background color is set to 0, and all other bits are set to 1 on a monochrome surface. Thus, if the current area background color is blue, all blue bits in the bit map are set to 0, and all other bits are set to 1. The 0 bits constitute the background of the primitive.

If the pattern is solid (PATSYM_DEFAULT or PATSYM_SOLID in the supplied pattern set), the following occurs:

- If color dithering is switched off, and the application is drawing a color pattern to a color surface, the color nearest the color specified is used.
- If color dithering is switched on, and the application is drawing a color pattern to a color surface, a combination of colors can be used to achieve the effect of the requested color. For example, if the application chooses pink on a surface where pink is not available, a combination of red and white pels can be used to achieve the effect of the color.
- If color dithering is switched on, and the application is drawing a color pattern to a monochrome surface, sufficient pels are set to suggest the intensity of the requested color.

Dithering can be enabled and disabled using LCOL_PURECOLOR in GpiCreateLogColorTable.

Using Color and Mix Attributes

The color- and mix-attribute functions can be used to perform the following tasks:

- Create a logical color table.
- Determine the format and the starting and ending index values of the current logical color table.
- Determine the index value for an entry in the logical color table that is the closest match to an RGB value.
- Determine the RGB value associated with a particular entry in the logical color table.
- Determine and set the current foreground and background colors.
- Determine and set the current foreground and background mix attributes.

Creating a Logical Color Table

To create a logical color table, the application creates an array of RGB values that replace the existing logical color table, then calls GpiCreateLogColorTable, using the LCOL_RESET and LCOLF_CONSECRGB flags. Figure 7-9 on page 7-21 demonstrates this process.

```

#define INCL_GPILOGCOLORTABLE
#include <os2.h>

void fncCOLR01(void){
    HPS hps;
    LONG aTable[] = {
        0xFFFFF, /* White */
        0xFF88FF,
        0xFF8800,
        0xFF8888,
        0xFF0088,
        0x880088,
        0x008888,
        0x00FF88,
        0x00F800,
        0x008800,
        0x000088,
        0x0000F8,
        0x0800F8,
        0x888888,
        0x080808,
        0x000000 }; /* Black */

    GpiCreateLogColorTable(hps,
        LCOL_RESET, /* Start with the default */
        LCOLF_CONSECRGB, /* Consecutive RGB values */
        0, /* Starting index in table */
        sizeof(aTable) / sizeof(LONG), /* Number of elements in table */
        aTable);
} /* fncCOLR01 */

```

Figure 7-9. Creating a Logical Color Table

Determining the Color-Table Format and Index Values

To determine the format and the starting and ending index values of the current logical color table, applications call `GpiQueryColorData`. Figure 7-10 is an example of using `GpiQueryColorData` to determine whether the default logical color table is loaded. If so, the code fragment loads a new table.

```

#define INCL_GPILOGCOLORTABLE
#include <os2.h>

void fncCOLR02(void){
    HPS hps;
    LONG aClrData[3];
    LONG aTable[16];

    GpiQueryColorData(hps, 3, aClrData);

    if (aClrData[QCD_LCT_FORMAT] == LCOLF_DEFAULT)
        GpiCreateLogColorTable(hps,
            LCOL_RESET, /* Start with the default */
            LCOLF_CONSECRGB, /* Consecutive RGB values */
            0, /* Starting index in table */
            sizeof(aTable) / sizeof(LONG), /* Number of elements in table */
            aTable);
} /* fncCOLR02 */

```

Figure 7-10. Determining Color-Table Format and Index Values

Determining the Index Value of an RGB Value

Applications call `GpiQueryColorIndex` to find the closest match in a logical color table to an RGB value. This function finds the closest match for this RGB value in the physical color table, then finds the color in the logical color table closest to the color in the physical color table. The function returns the index value of that entry in the logical color table. Figure 7-11 is an example of how to determine which index value in the logical color table matches the RGB value for pink (0x00FF00FF), then uses that index entry to set the foreground color to pink for each of the primitive attributes.

```
#define INCL_GPILOGCOLORTABLE
#include <os2.h>

void fncCOLR03(void){
    LONG lIndex;                                /* Logical-color-table index */
    HPS hps;

    lIndex = GpiQueryColorIndex(hps, LCOLOPT_REALIZED, 0x00FF00FF);

    if ((lIndex >= 0) && (lIndex <= 15))        /* Check for valid index */
        GpiSetColor(hps, lIndex);
    } /* fncCOLR03 */
```

Figure 7-11. Determining the Index Value of an RGB Value

Setting the Primitive Color Attributes

To set the color attributes for 1 type of graphics primitive, an application uses `GpiSetAttrs`. To set the color attributes for each type of primitive in a presentation space, use `GpiSetColor` and `GpiSetBackColor`. Figure 7-12 shows how to use `GpiSetAttrs` to set the color attribute of line primitives to dark gray.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>

void fncCOLR04(void){
    LINEBUNDLE lbind;    /* Line-primitive attribute structure */
    HPS hps;

    lbind.lColor = CLR_DARKGRAY;

    GpiSetAttrs(hps, PRIM_LINE, LBB_COLOR, 0, &lbind);
    } /* fncCOLR04 */
```

Figure 7-12. Setting Primitive Color Attributes

Figure 7-13 is an example of how to use `GpiSetColor` to set the foreground color attribute to dark gray in all of the primitives.

```
#include <os2.h>

void fncCOLR05(void){
    HPS hps;

    GpiSetColor(hps, CLR_DARKGRAY);
} /* fncCOLR05 */
```

Figure 7-13. Setting Foreground Color Attributes

Creating a Palette

To create a palette, an application must first call `DevQueryCaps` with the `CAPS_ADDITIONAL_GRAPHICS` option to determine whether the device supports palette functions. Next the application creates an array of RGB values or an array of `RGB2` structures, then calls `GpiCreatePalette`. Next the application calls `GpiSelectPalette` to select the palette for the presentation space, and calls `WinRealizePalette` to map the RGB values to device colors for subsequent drawing. When the application is finished drawing, it calls `GpiSelectPalette` to disassociate the presentation space and the palette; then it deletes the palette by calling `GpiDeletePalette`.

Figure 7-14 demonstrates these steps.

```
#define INCL_GPILOGCOLORTABLE
#define INCL_GPIBITMAPS
#include <os2.h>
void fncCOLR06(void){
    COLOR clrCurrent;
    HPAL hpal;
    HDC hdc;
    HPS hps;
    HAB hab;
    HWND hwnd;
    POINTL aptl[2], ptl;
    LONG cSimulColors, lPalSupport;
    SHORT j;
    RGB2 *prgb2ColorData;
```

Figure 7-14 (Part 1 of 2). Creating a Palette

```

/* Determine how many colors the device can display at once. */
DevQueryCaps(hdc, CAPS_COLORS, 1, &cSimulColors);

/* Determine if the device supports Palette Manager functions. */
DevQueryCaps(hdc, CAPS_ADDITIONAL_GRAPHICS, 1, &lPalSupport);

/* Allocate space for the array of RGB2 structures. */
DosAllocMem((PPVOID)&prgb2ColorData, cSimulColors * sizeof(RGB2), fALLOC);

/* Fill the array of RGB2 structures with as many different
/* shades of blue as the device will support. */

clrCurrent = 0x000000FF;
for (j = 0; j < cSimulColors; j++) {
    prgb2ColorData[j].bRed = 0;
    prgb2ColorData[j].bGreen = 0;
    prgb2ColorData[j].bBlue = clrCurrent;
    prgb2ColorData[j].fcOptions = 0;
    clrCurrent = clrCurrent > 0 ? --clrCurrent : 0x000000FF;
}

if (lPalSupport & CAPS_PALETTE_MANAGER) {
    hpal = GpiCreatePalette(hab, /* Create palette */
        0L,
        LCOLF_CONSECRGB, /* Format of color table entries */
        cSimulColors, /* Number of entries in table */
        (PULONG) prgb2ColorData); /* Pointer to color table */
}

GpiSelectPalette(hps, hpal);
WinRealizePalette(hwnd, hps);
GpiSelectPalette(hps, NULLHANDLE); /* Restore default physical colors */
GpiDeletePalette(hpal); /* Delete palette */
} /* fncCOLR06 */

```

Figure 7-14 (Part 2 of 2). Creating a Palette

Summary

Table 7-7 summarizes the color and mix attribute functions.

Table 7-7 (Page 1 of 2). Color and Mix Attribute Functions	
Function Name	Description
GpiAnimatePalette	Changes the color values of animating indexes in a palette.
GpiCreateLogColorTable	Loads a new logical color table.
GpiCreatePalette	Creates and initializes a color palette.
GpiDeletePalette	Deletes a color palette.
GpiQueryBackColor	Determines the current background color from the character string bundle attributes.
GpiQueryBackMix	Determines the current background mix attribute from the character string bundle attributes.
GpiQueryColor	Determines the current foreground color from the character string bundle attributes.
GpiQueryColorData	Determines information about the current logical color table.

Table 7-7 (Page 2 of 2). Color and Mix Attribute Functions

Function Name	Description
GpiQueryColorIndex	Determines the index value for the logical color table entry closest to a specified RGB value.
GpiQueryLogColorTable	Determines the current logical color table contents.
GpiQueryMix	Determines the current foreground mix attribute from the character string bundle attributes.
GpiQueryNearestColor	Determines the RGB value available on an output device, closest to a desired RGB value.
GpiQueryPalette	Determines the palette currently selected in a presentation space.
GpiQueryPaletteInfo	Determines the RGB, the index values, or both, for the currently loaded colors.
GpiQueryRealColors	Determines the RGB values for actual colors in the device color table.
GpiQueryRGBColor	Determines the RGB value paired with a logical color table index.
GpiRealizeColorTable	Maps the logical color table into the physical color table.
GpiSelectPalette	Selects a new palette for the presentation space.
GpiSetBackColor	Sets the background color for character strings, markers, areas, images, and bit maps.
GpiSetBackMix	Sets the background mix attribute for character strings, markers, areas, images, and bit maps.
GpiSetColor	Sets the foreground color for arcs, lines, character strings, markers, areas, images, and bit maps.
GpiSetMix	Sets the foreground mix attribute for arcs, lines, character strings, markers, areas, images, and bit maps.
GpiSetPaletteEntries	Changes entries in a logical palette.
GpiUnrealizeColorTable	Disassociates the mapped logical color table from the physical color table.

Table 7-8 summarizes the data structures used by the color and mix attribute functions.

Table 7-8. Color and Mix Attribute Structures

Structure Name	Description
AREABUNDLE	Area primitive attributes.
CHARBUNDLE	Character primitive attributes.
IMAGEBUNDLE	Image primitive attributes.
LINEBUNDLE	Line primitive attributes.
MARKERBUNDLE	Marker primitive attributes.
RGB	Red, green and byte color values.
RGB2	Same as RGB with the addition of an option-flag.

Chapter 8. Bit Maps

Raster output devices, such as display screens, are made up of a number of picture elements, called pixels or pels. By setting the color of the pels, you can create an image on the screen. The screen image can be represented internally by a graphics object called a *bit map*. The bit map contains a number of bits that describe the appearance of each pel on the screen.

This chapter describes bit maps, their creation, uses, and functions. The following topics are related to the information in this chapter:

- Presentation spaces and device contexts
- Coordinate spaces
- Color and mix modes
- Area primitives
- Paths.

About Bit Maps

Applications can use bit maps to:

- Store and display scanned images, icons, and symbols
- Create fill patterns for area primitives and paths.

An application can display a bit-map image on any *raster* output device. A raster is a rectangular matrix of pels on a video display or dot matrix printer. A raster output device displays an image by setting pels in its matrix to colors specified in a corresponding bit map. An image created in this way is called a "bit-map image." Bit maps cannot be sent to vector output devices such as plotters.

A bit map is drawn to an output device row-by-row. Each horizontal line of pels is known as a *scan line*.

There is a 1-to-1 correspondence between the number of rows of pels in a bit-map image and the rows of bits in a bit map. The first pel in a bit-map image is in the lower-left corner, and the last pel is in the upper-right corner. The pels are in left-to-right order inside each row of the image. Figure 8-1 shows this relationship between bit map and image.

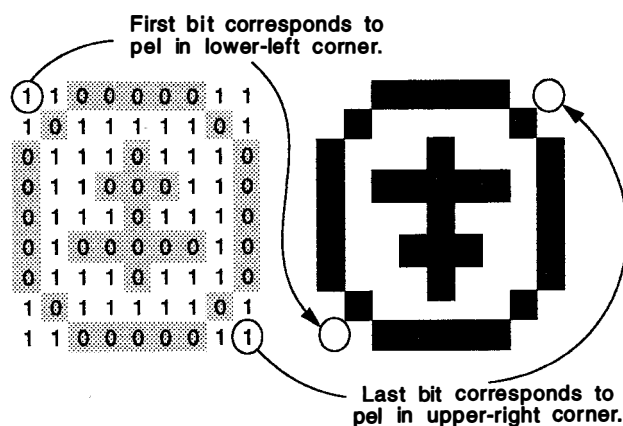


Figure 8-1. Bits and Pels in a Bit-Map Image

When an application creates a bit map by calling `GpiCreateBitmap`, it specifies the bit-map width and height in terms of pels in the bit-map image. The width is the number of pels within a row; the height is the number of rows. The application must store these dimensions in the `BITMAPINFO2` and `BITMAPINFOHEADER2` structures and pass their addresses to `GpiCreateBitmap`.

System Implementation

Bit maps are most useful when rapid and frequent movement is required, such as with icons and pointers. They are especially useful for restoring the contents of a window—for example, when an overlying window is removed. If you save the contents of a window in a bit map, you can restore the window contents simply by redisplaying the bit map when the window needs to be redrawn.

Using bit maps is also an effective method of erasing some of the screen contents. For example, you can save the image of the screen in a bit map at any time while drawing on the screen. If you continue drawing after saving the screen image, you can “erase” any drawing done since you saved the screen image by redisplaying the bit map.

Bit maps are not, however, the recommended way to store graphics that are going to be changed. Most changes to the bit-map contents mean that you have to re-create the bit map.

Bit-map images are device-dependent. Their appearance is affected by the shape of the device's pels and the device's color capabilities. For example, if the pels on one display measure 0.05 mm by 0.1 mm, but 0.1 mm by 0.3 mm on a second display, a circular bit-map pie chart drawn on the first display appears elliptical on the second. Figure 8-2 shows how a bit map appears on two types of displays.

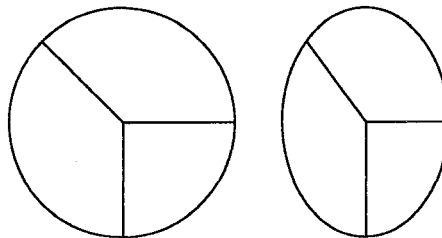


Figure 8-2. Bit Map Shown on Two Types of Displays

Bit maps, particularly color bit maps, can also occupy large amounts of memory. The actual amount of memory occupied by a bit map is determined by both the size of the bit map and the number of bits used to describe each pel.

Bit Map Functions

OS/2 2.0 provides a set of functions that allow you to:

- Create bit maps
- Create and load custom bit maps
- Store color information on a bit map
- Draw bit maps
- Transfer bit maps
- Change the size of a bit map
- Specify the mix values for a bit map
- Convert between monochrome and color data
- Manipulate single pels
- Copy images from a display into a bit map
- Save a bit map
- Delete a bit map
- Make a bit map available to other processes.

Creating a Bit Map

A bit map can be created by an application, or by using the PM Icon Editor.

By an Application: To enable an application to create a bit map:

1. Create a *memory device context*.

A memory device context enables an application to treat a bit map in memory as though it were a device. For example, an application can copy color information from another bit map, or copy pels on the display, into a bit map associated with a memory device context.

To create the memory device context, call DevOpenDC with:

- A device type of OD_MEMORY (second argument)
- A handle to a compatible device context (such as the device context of a device with which the bit map is to be compatible).

Note: The device device-context handle ideally should be the handle of the actual device to which you will be directing the bit map; otherwise, it will be necessary to change ownership to the appropriate device driver before the BitBlt operation (which copies the bit map from one presentation space to one associated with a screen device context). As a consequence, the image may appear distorted.

If you omit the handle of the compatible device context by specifying NULL, screen compatibility is assumed.

2. Create a graphics presentation space and associate it with the memory device context.

The operating system requires this association before the application can perform many of the bit map operations. The handle of this graphics presentation space is required as input to subsequent bit-map-creation and manipulation functions.

3. Create the bit map.

When an application creates a bit map, the handle of the presentation space that you have associated with the memory device context causes the bit map to be created in a format that is compatible with the memory device context.

The application also passes two structures: the *bit-map information header* and the *bit-map information table*. These structures contain a great deal of information about the bit map.

To create the bit map, call `GpiCreateBitmap` with:

- The handle of the presentation space (first argument)
- The bit-map information header, `BITMAPINFOHEADER2` (second argument).

The bit-map information header data structure is defined in the header files in the OS/2 2.0 Toolkit. The bit-map information table contains similar information, with the addition of the RGB2 array structure.

`GpiCreateBitmap` returns a handle to the bit map, which is used to identify the bit map.

To determine which bit-map formats are supported on a particular device, call `GpiQueryDeviceBitmapFormats`. This returns every format supported on a named device. The data is returned as an array of bit-map plane and bit-count pairs. The first pair of values in the array is the one most suitable for the device.

You can think of the bit map at this stage as a rectangular area of memory containing random data. You can initialize the bit map at this stage by providing `GpiCreateBitmap` with the address in application storage of some initialization data and by setting the `CBM_INIT` option. This is a useful function if, for example, your application always starts by displaying the same image.

4. Select the bit map.

Before selecting the bit map, you can disassociate the presentation space from the original memory device context and associate it with a different memory device context. However, the bit-map format must be convertible to a format that is supported by the new device. If you have selected one of the four standard bit-map formats, this compatibility is guaranteed and the conversion is automatic.

Note: When a presentation space is associated with a memory device context, a bit map must be selected into the device context before you can draw in the presentation space.

To select the bit map, call `GpiSetBitmap` with:

- The presentation-space handle (first parameter)
- The bit map-handle (second parameter).

Figure 8-3 shows the sequence of events when you create and display a bit map.

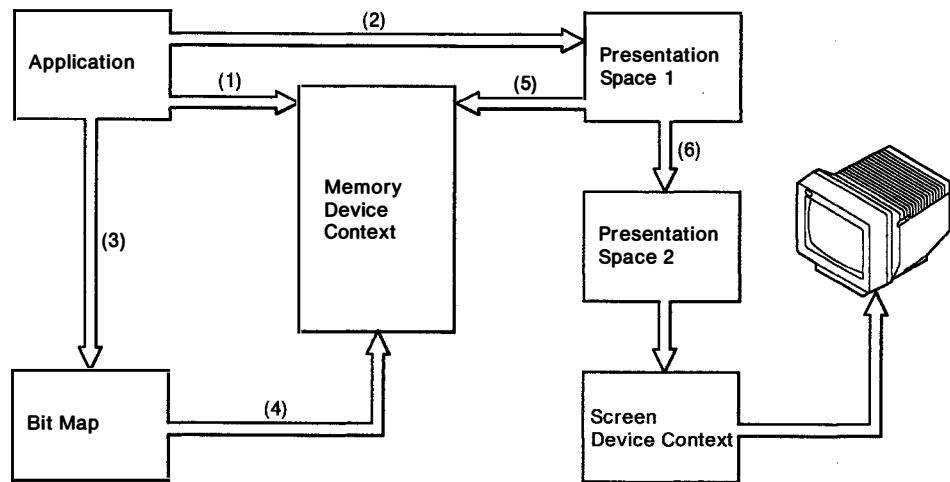


Figure 8-3. Creating and Displaying Bit Maps

The application:

1. Calls `DevOpenDC` to create the memory device context.
2. Creates a graphics presentation space. This is associated with the memory device context.
3. Calls `GpiCreateBitmap` to define a bit map.
4. Calls `GpiSetBitmap` to designate the bit map as the one currently selected in the memory device context.
5. Calls drawing instructions to the Presentation Space to draw to the bit map.
Note: If the bit map is initialized when it is created, this step does not normally exist. Alternately, this step can be a `GpiSetBitmapBits` call.
6. Calls `GpiBitBlt` to copy the bit map from Presentation Space 1 to Presentation Space 2 (associated with a screen device context). The bit map is transferred directly to the screen.

Using the Icon Editor: Using the Icon Editor, you can create monochrome or color bit maps that have a *static appearance*. This means that the bit maps can be created in advance and then used without change while the application is running.

When you use the Icon Editor to create a bit map, the bit map is saved in a resource file that can be loaded whenever it is needed. To load a bit-map file, call `GpiLoadBitmap`, with the identifier of the resource file that contains the bit map, as the second parameter. If you allow this value to default, the application's .EXE file is assumed to contain the bit map.

GpiLoadBitmap lets you specify the x- and y-dimensions (in pels) of the bit map. The loaded bit map is stretched or compressed accordingly. If you supply a 0 value for one of these dimensions, the bit map is sized in the other dimension only, which is likely to cause distortion of the image. If the bit map is to be produced in its original size, specify 0 for both its width and its height.

Output from the call to this function is the bit-map handle. To display the loaded bit map on the screen, follow the sequence of steps described in "Creating a Bit Map" on page 8-3, omitting steps 3 (defining the bit map) and 5 (issuing drawing instructions to the presentation space).

A bit map created by the Icon Editor is saved in a device-independent format. This format generates an array of bit maps with formats (bits per pel) matching each of the supported display devices.

Creating and Loading a Custom Bit Map

An application can create a custom bit map by setting the bits in an array and passing the array to GpiCreateBitmap, or by running the Icon Editor and loading the bit map with GpiLoadBitmap.

To create a custom bit map with an array, an application:

1. Defines an array of bytes that will set pels in an image to the appropriate colors. This array of bytes typically represents the output of a scanned image.
2. Sets the fields in the BITMAPINFOHEADER2 structure to their appropriate values.
3. Sets the fields in the BITMAPINFO2 structure to their appropriate values.
4. Calls GpiCreateBitmap, passes it the addresses of the structures and the array of bytes that the application has already defined, and sets the fIOptions parameter to CBM_INIT.

If the application is to use this bit map as a fill pattern, it assigns the bit map a local identifier by calling GpiSetBitmapId.

To load a custom bit map that was created with the Icon Editor:

1. Copy the bit map file to the directory in which you compile your application.
2. Create a BITMAP entry in your application's resource file, assigning a unique integer identifier to the bit map.
3. Call GpiLoadBitmap, passing it the identifier that you assigned to the bit map in the resource file.

An application can use GpiLoadBitmap to load any bit map from a file that conforms to any of the standard OS/2 bit-map formats or to a device-specific format supported by the device concerned. This means that an application can load a bit map created by another application, if that application created the correct bit-map header and stored the bit-map bits correctly.

Storing Color Information in a Bit Map

Graphics systems use one of two formats for storing color information in bit maps. The first format uses a single plane and a multiple bit count. The second format uses multiple color planes.

Color Planes: Bit maps are arranged in one or more *color planes*. A color plane is an array of bit-map bits that contains color information.

The bit maps in each of the previous illustrations use the single *bit-map plane* format, which is the standard format for bit maps in OS/2 applications. In this format, a specified number of adjacent bit-map bits contains indexes to either a special color table of RGB values or actual RGB2 structures. Whether the application maps bits into an RGB or RGB2 structure depends on the bit-map format. All of the color information resides in a single plane.

The second color format uses more than one color plane. A common multiplane bit-map format is the 3-plane format, in which one plane corresponds to the red pels, another to the green pels, and a third to the blue pels. Multiplane bit-map formats are rare in PM applications. Most bit maps are stored externally in a single-plane format, although the device driver (such as VGA) may internally convert them to the multiplane format.

The single-plane format can be converted internally to any multiplane format used by a device. You also can use a nonstandard number of bits to describe each pel, if supported by your output device. If you write your own presentation driver, it must be able to convert the standard bit-map formats to its own internal format.

An application can determine which color-plane format a device supports by calling `GpiQueryDeviceBitmapFormats`.

Standard Bit-Map Formats: On a monochrome device, you need only 1 bit to describe a single pel, and that bit is switched on or off. Color devices require more bits. For example, an 8-color picture requires 3 bits to describe a single pel, because each component of the RGB mix (red, green, blue) that gives a pel its color must be described.

A bit count is a value that specifies how many adjacent bit-map bits correspond to each pel in a bit-map image. There are four standard bit-map formats, each with a different bit count. The formats are shown in the following table.

Table 8-1. Standard Bit-Map Formats		
Format	Bits per pel	Size of 640x480 image in bytes (uncompressed)
Monochrome	1	38 400
16 color	4	153 600
256 color	8	307 200
16.7 million color	24	921 600
Note: The bits are stored consecutively in the bit-map plane. If you have 4 bits for each pel, the 4 bits for pel 1 are followed by the 4 bits for pel 2, and so on. The bits that describe pel 1 are stored beginning in the most-significant bits of the first byte. The data for each scan line is packed together, and the bottom scan line appears first in memory with the leftmost pel first. Each scan line, however, is padded at the end so that each line begins on a ULONG (32-bit) boundary.		

If the device supports a bit count of 1 bit per pel, the color table contains 2 entries. A device that supports a bit count of n bits per pel, has a corresponding color table with 2^n entries. However, a bit count of 24 bits per pel indicates that there is no color table, because each pel is a direct RGB value.

Figure 8-4 shows a bit map using a bit count of 4 bits per pel and an associated color table:

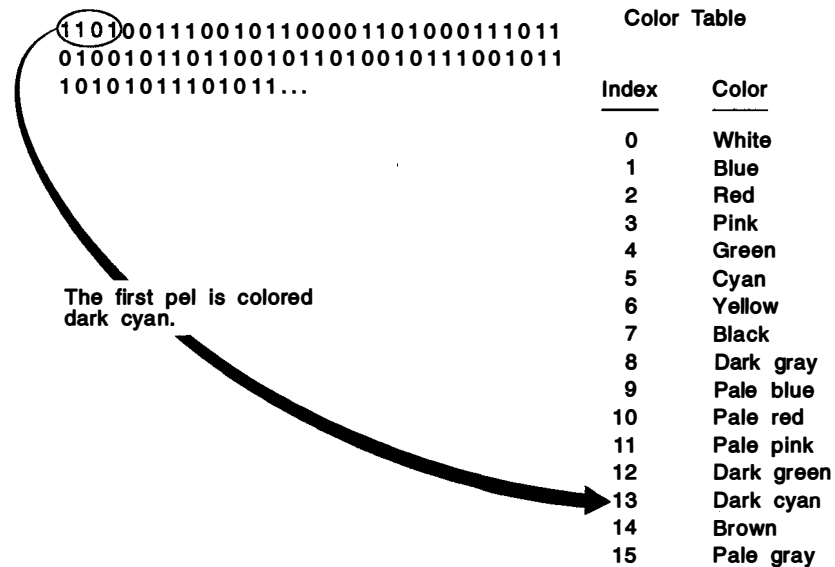


Figure 8-4. A Bit Map and Its Associated Color Table

If a device uses a bit count of 1, 4, or 8 bits per pel, the bit-map bits contain index values for a bit-map color table. If the device supports a bit count of 24 bits per pel, the bit-map bits contain the bRed, bGreen, and bBlue fields of RGB2 structures. No color table is associated with a bit map on a device that supports a format of 24 bits per pel—such a device can support over 16 million colors. Instead of using a color table, the BITMAPINFO2 structure consists of only the header, and the red, green, and blue color values are provided directly by the bit-map data.

An application can determine the bit-count format that a device supports by calling GpiQueryDeviceBitmapFormats.

Drawing Bit Maps

An application can draw bit-map images on a raster printer or display screen, or into metafiles associated with a raster device. Any GPI drawing requests (including those that produce graphics text), issued to a presentation space associated with a memory DC containing a selected bit map, cause the bit map to receive raster images of your drawings. Table 8-2 on page 8-9 describes the bit map drawing functions:

Table 8-2. Bit-Map Drawing Functions

Function	Input	Output
WinDrawBitmap	The handle of a bit map.	A bit-map image on a raster display.
GpiImage	A buffer containing bit map image data.	A special monochrome bit-map image on a raster display or printer.
GpiDrawBits	A buffer containing bit map image data.	A bit-map image on a raster display or printer.
GpiBitBit	The handle of a presentation space containing a bit map.	A bit-map image on a raster display or printer, or a bit-map image into a metafile (albeit in an escape order).
GpiWCBitBit	The handle of a bit map.	A bit-map image on a raster display or printer, or a bit-map image into a metafile.

WinDrawBitmap: WinDrawBitmap draws a bit-map image by copying it into a window linked to a target presentation space. A call to this function is valid only in draw mode (DM_DRAW), and only for a screen device context. This function does not require an application to select a bit map into a presentation space before the application draws the corresponding image. An application can use WinDrawBitmap to scale bit maps by specifying DBM_STRETCH as the last argument, and the address of a RECTL structure as the fourth argument. The coordinates in this structure are always device coordinates.

WinDrawBitmap draws both color and monochrome bit maps. Color bit maps require no color conversion. Monochrome bit maps can be drawn in any two colors which can be explicitly specified as parameters to the call or taken from the image bundle. These parameters will be color table indexes or RGB values, depending on the color table mode of the target presentation space. The current image bundle mix modes are used and, for certain mix values, will affect color.

You can call WinDrawBitmap in retain mode, but the bit-map image will only be drawn and not recorded in the segments.

Note: An application can determine the current colors and their corresponding mix modes by calling GpiQueryAttrs. The application can set them by calling GpiSetAttrs.

An *inverted* bit map is a bit map in which the colors have been inverted; white becomes black and black becomes white. An application can draw inverted bit maps by calling WinDrawBitmap and passing it DBM_INVERT as the last argument. An application can draw halftone bit maps by calling WinDrawBitmap and passing it DBM_HALFTONE as the last argument. Before drawing the bit map, clear the presentation space to CLR_BACKGROUND using GpiErase.

GpiImage: GpiImage draws a nonstandard, monochrome (two-color) bit map called an *image primitive*. The bit-map bits in an image are stored in the opposite order from the bits in a standard bit map—the first bit in the bit map corresponds to the pel in the upper-left corner of the bit-map image, and the last bit in the bit map corresponds to the pel in the lower-right corner of the bit-map image. Figure 8-5 on

page 8-10 shows the correspondence between the bits in an image primitive and the pels in the drawing produced by GpImage.

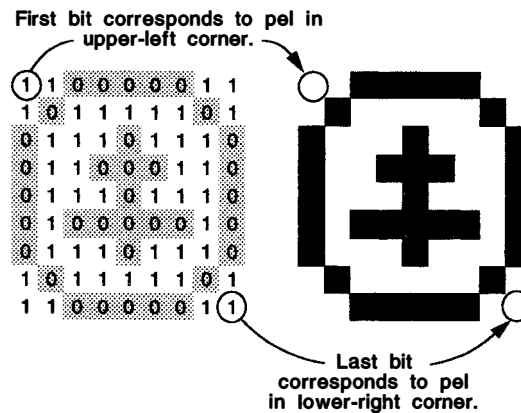


Figure 8-5. Image-Primitive Bits and Pels

A call to GpImage also is valid only in draw mode (DM_DRAW), but can provide output to a screen or printer. An application cannot scale bit-map images by using GpImage.

Transferring Bit Maps

The three remaining bit-map drawing functions operate in very similar ways. GpiDrawBits copies a bit-map image from application memory to a device or a device context; GpiBitBlt directs bit maps to devices other than the screen; GpiWCBitBlt enables you to retain the bit-map data in the segment store of the target presentation space.

The similarities are discussed first. Their differences are discussed in the sections that follow.

An application should use GpiDrawBits, GpiBitBlt, or GpiWCBitBlt to draw bit maps that use a color table or RGB2 structures color formats of 1, 4, 8, or 24 bits per pel.

An application can draw inverted bit maps for any of these three functions by calling the function and passing ROP_NOTSRCCOPY as the raster operation.

GpiDrawBits: GpiDrawBits copies bit map image data from storage into a bit map that has been selected into a device context associated with a presentation space. It can also copy bit-map image data to a device. An application can use this function to draw a bit map without first selecting the bit map into a presentation space.

This function is valid in all draw modes. Set the draw mode to DM_RETAIN or DM_DRAWANDRETAIN to create retained segments; otherwise, the default mode, DM_DRAW, is selected.

GpiBitBlt: GpiBitBlt requires an application to use device coordinates for the dimensions of the source and target rectangles. GpiBitBlt allows you to direct bit maps to devices other than the screen. It is independent of the drawing mode, but it operates as if in draw mode.

At its simplest, GpiBitBlt copies a whole bit map, unaltered, from a device or bit map source to a device or bit map target. At its most powerful, GpiBitBlt can take a part of or a whole bit map, and alter both its size and appearance in the process of copying it to another device context. A bit map can be copied:

- From one memory device context to another memory device context
- From a memory device context to the device context of an output device (screen window or printer) that supports raster operations
- From the device context of an output device to a memory device context
- From the device context of an output device to another device context of the same output device. In both cases, the screen device can be a PM window.

A memory device context (whether it is the source or the target of the GpiBitBlt operation) *must* have a bit map selected when GpiBitBlt is called.

GpiBitBlt has a number of input parameters, two of which are the handles of two presentation spaces: the source presentation space and the target presentation space. Both of these presentation spaces must be associated with an appropriate device context. Unless the associated device is a banded printer, a single presentation space can be both source and target. This allows you to copy a bit map within a single PM window, or to update the bit map. The source and target rectangles are specified in device coordinates in GpiBitBlt. GpiBitBlt, consequently, is very device-dependent, and should be avoided when creating data for interchange.

GpiWCBitBlt: GpiWCBitBlt enables you to retain the bit-map data in the segment store of the target presentation space. It is valid in all three drawing modes:

- DM_DRAW—display, printer, or into metafile
- DM_RETAIN—into metafile or segment
- DM_DRAWANDRETAIN—display, printer, then into associated segment or metafile.

Most of the bit-map drawing operations occur in an application's device space; GpiWCBitBlt, however, lets an application draw in its world space, but requires that you use device coordinates for the source rectangle, and world coordinates for the target rectangle.

An application can use GpiWCBitBlt to draw a bit map with consistent dimensions on devices with different *aspect ratios*. The aspect ratio is the ratio of a pel's width to its height.

When creating data for interchange, use GpiWCBitBlt. GpiWCBitBlt provides the same function as GpiBitBlt, with the following exceptions:

- The target rectangle is specified in world coordinates, and all four coordinates (the two source-rectangle coordinates and the two target-rectangle coordinates) must be specified.
- The source handle must be a bit-map handle. It must not be the handle of a source presentation space. The bit map identified by the source handle must not be selected into a memory device context when you call GpiWCBitBlt.
- GpiWCBitBlt conforms to the current drawing mode in the target presentation space. If the drawing mode is retain or draw-and-retain, the bit map is retained in segment store.

Changing the Size of the Bit Map

You can specify the sizes of the rectangular bit blocks in both the source and the target presentation spaces. To do this, provide an array of up to four device-coordinate positions as input to the call. The first two positions define the lower-left and upper-right corners of the target rectangle; the second two define the same two corners of the source rectangle.

If you want the two rectangles to be of equal size, do not specify the device coordinates of the upper-right corner of the source rectangle. The correct amount of data is automatically transferred to fill the target rectangle. Figure 8-6 shows the points count for bit-block transfers.

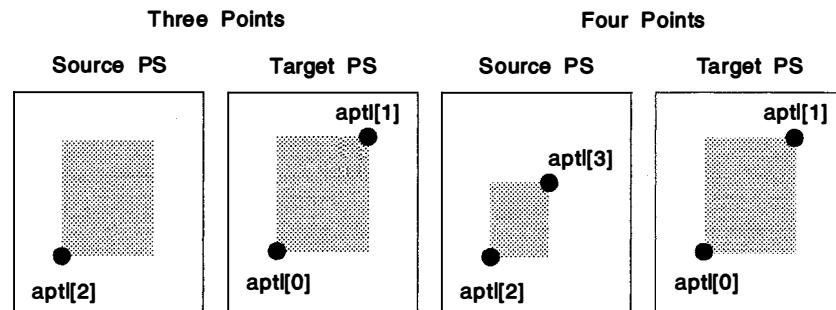


Figure 8-6. Points Count for Bit-Block Transfers. Equal-size rectangles can be built much faster than rectangles that need stretching or compressing. Compression options (flOptions) are ignored if the rectangles are to be of equal size.

If the rectangles are not to be of equal size, you must specify all four coordinate points. The bit-map data is stretched, if the target rectangle is larger than the source rectangle, or compressed, if the target rectangle is smaller, to fit the target rectangle. The bit map is stretched by duplicating rows and columns of data, an action that might cause distortion of the image. If the data is to be compressed, you can specify one of three compression options as shown in Table 8-3.

Table 8-3. Bit-map Data Compression Rules	
Option	Compression Rules
BBO_OR	Compresses the bit map data as necessary, using a logical OR operation on the eliminated rows and columns. This is useful for preserving the foreground when foreground pels are "1" and the background pels are "0".
BBO_AND	Compresses the bit map data as necessary, using a logical AND operation on the eliminated rows and columns. This is useful for preserving the foreground when foreground pels are "0" and the background pels are "1".
BBO_IGNORE	Compresses the bit map data as necessary, but ignores any eliminated rows or columns. This is most useful for color bit maps, where the results of combining pels of different colors are unpredictable.

Specifying Mix Values

When you draw a graphics primitive into a presentation space, it is affected by the current foreground-mix value, and possibly by the current background-mix value, in that presentation space. However, when you copy a bit map into a target presentation space, the current foreground- and background-mix values in the target presentation space are ignored. Instead, you specify a mix-mode value, also known as the *raster-operation* (ROP) value. This value determines:

- How the bit map is affected by any data that might already be in the target presentation space
- The color of each pel in the bit map.

Each pel in the bit map of the target presentation space has, potentially, three color settings:

- The setting of that pel in the source rectangle
- The initial setting of that pel in the target rectangle
- The setting of that pel in the current area-fill pattern in the target presentation space.

The (boolean) values of each of these settings can be combined using boolean operations to produce the final value of each pel in the target presentation space. For a color target, the target must be regarded as consisting of multiple one-bit per pel planes, with the mixing applied to each plane separately to produce the final color index into the physical color table. Because the final color index is an index into the physical palette, the results of any color mixing are therefore unpredictable. For example, if you ORed together two numeric index values, the color indexed by the result is unlikely to have any direct relation to the colors indexed by the two values you ORed together. It depends on the order of the colors in the physical table.

As input to the functions, you supply an actual mix value. The ROP mix value required to achieve any given result can be determined from Table 8-4 on page 8-14. The final value of each bit in every pel depends on the values of the corresponding bits in the pattern (P), source (S), and the original target value (T initial). Each row of the table shows one of the eight possible combinations of these values. For each combination, mark the desired final target value in the last column. The eight bits in this column then show the value of the least significant byte of the ROP value required to achieve this mixing function. For example, if the required mixing function is to copy the source to the target, then the Target (final) column will be the same as the S column, and the ROP value will have the binary value 11001100, or the hexadecimal value 00CC.

<i>Table 8-4. Possible Settings for the Index Bits</i>			
Pattern	Source	Target (Initial)	Target (final)
0	0	0	Index bit 0
0	0	1	Index bit 1
0	1	0	Index bit 2
0	1	1	Index bit 3
1	0	0	Index bit 4
1	0	1	Index bit 5
1	1	0	Index bit 6
1	1	1	Index bit 7

If you want the pattern to have no effect on the target rectangle, the four low-order bits of the index must be the same as the four high-order bits.

A monochrome bit map consists only of Source settings of 1s and 0s, where a 1 or 0 represents each pel. It also has a two entry color table with index 0 specifying what color a 0 pel represents and index 1 representing what color a 1 pel represents. In a monochrome bit map, this color table is not used during BitBlt operations (but is used when drawing to the bit map and moving the bit map between devices). Table 8-4 defines how any combination of pattern, source and target original (1 and 0) values define the target final (1 or 0) value. A color bit map has an index (in the case of 4 bits, 0-15) for each pel which, serves as index into the bit map color table (16 entries for 4 bits per pel) to define the color of each pel. Unlike monochrome bit maps, this table is used during BitBlt operations (instead of target image bundle attributes) to provide the color of the target output.

Table 8-5 represents a selection of the most commonly used mixes with predefined symbolic names (but there are many other possible mixes).

<i>Table 8-5 (Page 1 of 2). Mix Value Names</i>	
Mix Name	Effect
ROP_SRC COPY	Source
ROP_SRC PAINT	Source OR Target
ROP_SRC AND	Source AND Target
ROP_SRC INVERT	Source XOR Target
ROP_SRC ERASE	Source AND NOT (Target)
ROP_NOT SRCCOPY	NOT (Source)
ROP_NOT SRCERASE	NOT (Source) AND NOT (Target)
ROP_MER GE COPY	Source AND Pattern
ROP_MER GE PAINT	NOT (Source) OR Target
ROP_PAT COPY	Pattern
ROP_PAT PAINT	NOT (Source) OR Pattern OR Target
ROP_PAT INVERT	Target XOR Pattern
ROP_DST INVERT	NOT (Target)
ROP_ZERO	0
ROP_ONE	1

Table 8-5 (Page 2 of 2). Mix Value Names

Mix Name	Effect
ROP_GRAY	Gray pattern background

ROP_SRCCOPY is the most often used value. It simply copies the bit map to the target presentation space without performing any color mixing. Because most of the remaining mixes involve some degree of color mixing, they result in an image that is quite different from the original.

Converting between Monochrome and Color Data Bit Maps

It is possible to copy a monochrome bit map to a color bit map or device, and to copy a color bit map to a monochrome bit map or device. PM handles these conversions automatically according to the following rules:

- If you are copying a monochrome bit map to a color bit map or to a color or monochrome device surface, a Source setting of 1 adopts the current image-foreground color of the target presentation space, and a Source setting of 0 adopts the current image-background color of the target presentation space. For example, if the image foreground color is blue and the image background color is yellow in the target presentation space, a monochrome bit map is converted to a blue foreground on a yellow background.
- If you are copying from a monochrome pattern to a color bit map or device, a Source setting of 1 adopts the current area-foreground color of the target device and a Source setting of 0 adopts the current area-background color of the target device. Note that if image bundle attributes do not exist for the source bit map, as is the case when using GpiWCBtBit (which uses a bit map handle as the source), then zero source pixels adopt the image background color and nonzero source pixels adopt the image-foreground color of the presentation space.
- If you are copying a color bit map to a monochrome bit map or device, those pels that have the same color as the current image-background color in the *source* presentation space adopt the image background color of the *target* presentation space. For example, if the current image-background color in the source presentation space is blue, all blue pels in the bit map take on the color of the current image background in the target presentation space.

All other pels in the color bit map adopt the current image-foreground color of the *target* presentation space.

Manipulating Single Pels

You can manipulate single pels in a bit map by using the GpiSetPel and GpiQueryPel functions. GpiSetPel sets the pel at the specified point (in world coordinates) to the current line color. This function is independent of the current drawing mode, and its effect is immediate. It is, however, a device-dependent function. The CAPS_RASTER_CAPS option of the DevQueryCaps function indicates if GpiSetPel is supported on the current device.

Copying Images from a Display into a Bit Map

An application can copy an image from a raster display screen to a bit map by calling GpiBitBlit or GpiWCBtBit. Before copying the bit map, the application must call DevOpenDC to create a memory device context. This device context allows an application to treat a bit map in memory as though it were a device—the application can copy color information from pels on the display to the bit map.

Once an application creates a memory device context, associates it with a presentation space, and selects a bit map into the presentation space, the application can use the presentation-space handle as the first argument to `GpiBitBlt` or `GpiWCBitBlt`. If the application will be drawing the image on devices with different aspect ratios, the application should use `GpiWCBitBlt` to preserve the original dimensions of the bit map.

Saving a Bit Map

You have two ways to save a bit map that has been created by an application:

- Store the bit map in a *metafile*, by calling `GpiWCBitBlt`.
Metafiles are covered in detail later.
- Use the OS/2 operating system file-handling functions, in conjunction with `GpiQueryBitmapBits` and `GpiSetBitmapBits`.

An application can save a bit map in a file by calling `GpiQueryBitmapBits`, `DosOpen`, `DosWrite`, and `DosClose`.

`GpiQueryBitmapBits` copies bit-map data from a memory device context (that has a standard-format bit map selected into it) to a buffer. The bit map you copy can be newly created or have been loaded from a resource file.

You must supply the address in application storage of a *bit-map information table*. The bit-map information table has the same structure as the bit map information header, except for an additional entry, the colors field. As input to `GpiQueryBitmapBits`, you supply a bit-count value and a plane value. These must both be standard bit-map format values, so the plane value is always 1, and the bit-count value is 1, 4, 8, or 24. If any conversion of the bit-map data is necessary, it is done automatically. You also must provide the address of the application storage into which the bit-map data is to be loaded. You probably will need to find out how large the bit map is before you can do this. The `GpiQueryBitmapParameters` function returns the width, height, plane count, and bit count of a named bit map.

You also must supply the size of the fixed portion of the header (as well as bit count, and so on). Nonstandard formats supported by the device that owns the bit map will also be valid for `GpiQueryBitmapBits`.

You must ensure that you allocate sufficient storage at the end of the bit map information table for the returned color table which, if the number of planes is 1, will have 2^n entries, where n is the number of bits per pel.

On return from `GpiQueryBitmapBits`, the system supplies the width, height, and bit-map color table in the bit-map information table. You can retrieve a part of a bit map by specifying the number of the scan line from which the transfer is to start, and the number of scan lines you want.

When you have copied a bit map into application storage, you can save the bit map externally, and reload it later, by using OS/2 file-handling functions. After the application creates a file by calling `DosOpen`, it can call `DosWrite` to copy the buffer containing the bit-map bits into the file. The application then closes the file by calling `DosClose`.

If an application needs to use a bit map it has stored on disk, it can copy the file's contents into a buffer by calling `DosRead`, and then copy information about the image to the bit map by calling `GpiSetBitmapBits`. The application must select the bit map into a memory device context before it sets the bits. As with

GpiQueryBitmapBits, you can transfer a part of the bit map rather than the whole. GpiSetBitmapBits can be used to transfer standard-format bit maps only, and only to a bit map selected into a memory device context. If necessary, bit-map data is automatically converted from one standard format to another. Nonstandard formats supported by the device owning the bit map will also be valid for GpiSetBitmapBits.

If an application creates bit maps another application might use, it should create them by using the standard OS/2 operating system bit-map file format.

Deleting a Bit Map

It is good practice always to delete a bit map from memory when you have finished using it. To delete a bit map, use GpiDeleteBitmap. If you have loaded a bit map from a resource file, GpiDeleteBitmap deletes only its memory version. If the bit map has not been deleted when the owning process ends, it is deleted automatically by the system.

Making Bit Maps Available to Other Processes

When an application creates a bit map or loads one from a resource file, it can make the bit map available to other processes by placing the bit-map handle in the clipboard.

Using Bit Maps

An application can use bit-map functions to:

- Copy an image from a raster device (such as a display screen) to a bit map
- Copy an image from a raster device (such as a display screen) to the same device
- Copy an image from a bit map to a raster device
- Copy an image from a bit map to another bit map
- Copy an image from application memory to a bit map
- Copy an image from application memory to a raster device
- Scale bit-map images
- Create custom fill patterns for area primitives and paths
- Load a bit map created with the Icon Editor
- Draw bit-map images
- Store bit maps in a metafile or application memory.

Copying an Image from a Display Screen to a Bit Map

To copy an image from a display screen to a bit map:

1. Associate the memory device context with a presentation space.
2. Create a bit map.
3. Select the bit map into the memory device context by calling GpiSetBitmap.
4. Determine the location (in device coordinates) of the image.
5. Call GpiBitBlt and copy the image to the bit map.

Figure 8-7 demonstrates these steps.

```
HDC hdcMem;
PSZ pszData[4] = { "Display", NULL, NULL, NULL };
HPS hpsMem, hps;
HAB hab;
SIZEL sizlPage = {0, 0};
BITMAPINFOHEADER2 bmp;
PBITMAPINFO2 pbmi;
HBITMAP hbm;
SHORT sWidth = 128, sHeight = 128;
POINTL aptl[3];
LONG alData[2];

/*
 * Create the memory device context and presentation space so they
 * are compatible with the screen device context and presentation space.
 */

hdcMem = DevOpenDC(hab, OD_MEMORY, "", 4,
(PDEVOPENDATA) pszData, NULLHANDLE);

hpsMem = GpiCreatePS(hab, hdcMem, &sizlPage,
PU_PELS | GPIA_ASSOC | GPIT_MICRO);

/* Determine the device's plane/bit-count format. */
GpiQueryDeviceBitmapFormats(hpsMem, 2, alData);

/*
 * Load the BITMAPINFOHEADER2 and BITMAPINFO2 structures. The sWidth and
 * sHeight fields specify the width and height of the destination
 * rectangle.
 */

bmp.cbFix = (ULONG) sizeof(BITMAPINFOHEADER2);
bmp.cx = sWidth;
bmp.cy = sHeight;
bmp.cPlanes = alData[0];
bmp.cBitCount = alData[1];
bmp.ulCompression = BCA_UNCOMP;
bmp.cbImage = (((sWidth *
(1 << bmp.cPlanes) * (1 << bmp.cBitCount)) + 31) / 32) * sHeight;
bmp.cxResolution = 70;
bmp.cyResolution = 70;
bmp.cclrUsed = 2;
bmp.cclrImportant = 0;
bmp.usUnits = BRU_METRIC;
bmp.usReserved = 0;
bmp.usRecording = BRA_BOTTOMUP;
bmp.usRendering = BRH_NOTHALFTONED;
bmp.cSize1 = 0;
bmp.cSize2 = 0;
bmp.ulColorEncoding = BCE_RGB;
bmp.ulIdentifier = 0;
```

Figure 8-7 (Part 1 of 2). Copying a Display-Screen Image to a Bit Map

```

DosAllocMem((PPVOID)&pbmi, sizeof(BITMAPINFO2) +
    (sizeof(RGB2) * (1 << bmp.cPlanes) * (1 << bmp.cBitCount)),
    PAG_COMMIT | PAG_READ | PAG_WRITE);

pbmi->cbFix = bmp.cbFix;
pbmi->cx = bmp.cx;
pbmi->cy = bmp.cy;
pbmi->cPlanes = bmp.cPlanes;
pbmi->cBitCount = bmp.cBitCount;
pbmi->ulCompression = BCA_UNCOMP;
pbmi->cbImage = ((sWidth+31)/32) * sHeight;
pbmi->cxResolution = 70;
pbmi->cyResolution = 70;
pbmi->cclrUsed = 2;
pbmi->cclrImportant = 0;
pbmi->usUnits = BRU_METRIC;
pbmi->usReserved = 0;
pbmi->usRecording = BRA_BOTTOMUP;
pbmi->usRendering = BRH_NOTHALFTONED;
pbmi->cSize1 = 0;
pbmi->cSize2 = 0;
pbmi->ulColorEncoding = BCE_RGB;
pbmi->ulIdentifier = 0;

/* Create a bit map that is compatible with the display. */
hbm = GpiCreateBitmap(hpsMem, &bmp, FALSE, NULL, pbmi);

/* Associate the bit map and the memory presentation space. */
GpiSetBitmap(hpsMem, hbm);

/* Copy the screen to the bit map. */
aptl[0].x = 0; /* Lower-left corner of destination rectangle */
aptl[0].y = 0; /* Lower-left corner of destination rectangle */
aptl[1].x = sWidth; /* Upper-right corner of destination rectangle */
aptl[1].y = sHeight; /* Upper-right corner of destination rectangle */
aptl[2].x = 0; /* Lower-left corner of source rectangle */
aptl[2].y = 0; /* Lower-left corner of source rectangle */

hps = WinGetScreenPS(HWND_DESKTOP);

GpiBitBlt(hpsMem, hps,
    sizeof(aptl) / sizeof(POINTL), /* Number of points in aptl */
    aptl, ROP_SRCCOPY, BBO_IGNORE);

WinReleasePS(hps);

```

Figure 8-7 (Part 2 of 2). Copying a Display-Screen Image to a Bit Map

Scaling and Drawing a Bit-Map Image

You can scale a bit map by calling `GpiBitBlt` or `GpiWCBitBlt` and altering the dimensions of the target rectangle. Figure 8-8 shows how to shrink the screen copied in the first example to half its original size, and then redraw it by calling `GpiBitBlt`.

```
POINTL aptl[3];
HPS hpsMem, hps;
HWND hwnd;
SHORT sWidth = 128, sHeight = 128;
/* Target-rectangle dimensions (in device coordinates) */
aptl[0].x = 0;
aptl[0].y = 0;
aptl[1].x = sWidth / 2;
aptl[1].y = sHeight / 2;

/* Source-rectangle dimensions (in device coordinates) */
aptl[2].x = 0;
aptl[2].y = 0;
aptl[3].x = sWidth;
aptl[3].y = sHeight;

hps = WinGetPS(hwnd);

GpiBitBlt(hps, hpsMem,
          sizeof(aptl) / sizeof(POINTL), /* Number of points in aptl */
          aptl, ROP_SRCCOPY, BBO_IGNORE);

WinReleasePS(hps);
```

Figure 8-8. Scaling and Drawing a Bit-Map Image

Creating a Custom Fill Pattern

To create a custom fill pattern that the operating system will use to fill area primitives and paths:

1. Set an array of bits for a bit map that measures 8-bits-by-8-bits (remember that the operating system pads the bit-map bits on a ULONG (32-bit) boundary).
2. Create a bit map in a screen presentation space by calling `GpiCreateBitmap` and passing it the address of the array of bits from Step 1.
3. Assign a local identifier (`lcid`) to the bit map by calling `GpiSetBitmapId`.
4. Set the attribute of the pattern set in the `AREABUNDLE` structure by calling `GpiSetPattern`.

Figure 8-9 shows how to create the pattern.

```
/* Define an array of bytes; this array creates a grid pattern. */
BYTE abPattern5[] = {
    0xFF, 0xFF, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00 };

    LONG lcidCustom = 1;
    HPS hps;
    PBITMAPINFO2 pbmi;
    BITMAPINFOHEADER2 bmp;
    HBITMAP hbm;
    PRGB2 prgb2;

    /* Create the bit map, passing the address of the array of bytes. */
    hbm = GpiCreateBitmap(hps, &bmp, CBM_INIT, (PBYTE) abPattern5, pbmi);

    /* Assign a local identifier to the bit map. */
    GpiSetBitmapId(hps, hbm, lcidCustom);

    /* Set the pattern-set attribute in the AREABUNDLE structure. */
    GpiSetPatternSet(hps, lcidCustom);
```

Figure 8-9. Creating a Custom Fill Pattern

Loading a Bit Map from a File

You can load a bit map from a file if the format of the file corresponds to the standard OS/2 operating system bit-map file format. (Any bit map that you create with the Icon Editor is automatically stored in this format.) To load a bit map:

1. Copy the bit-map file to the directory that contains your application's resource file and source code.
2. Create an entry in your application's resource file, assigning a unique integer identifier to the bit map.
3. Call `GpiLoadBitmap` in your application's source code, passing it the integer identifier that you assigned to the bit map in your application's resource file.

You can actually include your bit map in the application's .EXE file or in a separate resource file. If the bit map is included in a separate resource file, you must specify both the resource ID and the ID of the bit map within the resource file on `GpiLoadBitmap`. If bit map is to be included in the application's .EXE file, the resource ID is specified as `NULL` and only the ID of the bit map is required by `GpiLoadBitmap`.

Following is an example of code from an application's resource file that assigns the integer value, 200, to a bit-map file called CUSTOM.BMP.

```
BITMAP 200 CUSTOM.BMP
```

Figure 8-10 is an example of code from the application that shows how to retrieve a bit-map handle by calling `GpiLoadBitmap`, use the handle to tag the bit map by calling `GpiSetBitmapId`, and then use the local identifier supplied by `GpiSetBitmapId` to set the bit map as the current fill pattern, using `GpiSetPatternSet`.

```
HPS hps;
HBITMAP hbm;
LONG lcidCustom = 1;
POINTL ptl;

hbm = GpiLoadBitmap(hps, /* Presentation-space handle */
    NULLHANDLE, /* Resource in application's module */
    IDB_PATTERN, /* Bit-map ID */
    16, /* Bit-map width */
    16); /* Bit-map height */

/* Assign a local identifier to the bit map. */
GpiSetBitmapId(hps, hbm, lcidCustom);

/* Set the pattern-set attribute in the AREABUNDLE structure. */
GpiSetPatternSet(hps, lcidCustom);

ptl.x = 100;
ptl.y = 100;
GpiMove(hps, &ptl);
ptl.x = 200;
ptl.y = 200;
GpiBox(hps, DRO_OUTLINEFILL, &ptl, 0, 0);
```

Figure 8-10. Loading a Bit Map from a File

Storing a Bit Map in a Metafile

You can draw bit maps in a metafile or segment by calling `GpiWCBitBlt`. The operating system converts this function to a drawing order. The target-rectangle dimensions that you pass to `GpiWCBitBlt` are in world coordinates, not device coordinates. Figure 8-11 shows how to draw a bit map in a metafile, and then play the metafile.

```

DEVOPENSTRUC dop;
HDC hdcMeta;
HPS hps, hpsMeta;
SIZEL sizlPage;
HMF hmf;
HBITMAP hbm;
HAB hab;
HWND hwnd;
POINTL aptl[4];

dop.pszLogAddress = NULL;
dop.pszDriverName = "DISPLAY";
dop.pdriv = NULL;
dop.pszDataType = NULL;

hdcMeta = DevOpenDC(hab, OD_METAFILE, "", 4L,
                    (PDEVOPENDATA) &dop, NULLHANDLE);
hpsMeta = GpiCreatePS(hab, hdcMeta, &sizlPage, PU_PELS | GPIA_ASSOC);

hbm = GpiLoadBitmap(hpsMeta, NULLHANDLE, IDB_PATTERN, 16L, 16L);

aptl[0].x = aptl[0].y = 0; /* Lower-left corner target rectangle */
aptl[1].x = 150;           /* X coordinate upper-right target rectangle */
aptl[1].y = 300;           /* Y coordinate upper-right target rectangle */
aptl[2].x = aptl[2].y = 0; /* Lower-left corner source rectangle */
aptl[3].x = aptl[3].y = 16; /* Upper-right corner source rectangle */

GpiWCBitBlt(hpsMeta, hbm, 4L, aptl, ROP_SRCCOPY, BBO_IGNORE);

GpiAssociate(hpsMeta, NULLHANDLE);
hmf = DevCloseDC(hdcMeta);

hps = WinGetPS(hwnd);

GpiPlayMetaFile(hps, hmf, 0L, NULL, NULL, 0L, NULL);

WinReleasePS(hps);

```

Figure 8-11. Storing a Bit Map in a Metafile

You can also store a bit map in a metafile by calling `GpiBitBlt`. In this case, however, the bit map will be stored inside an escape order.

Summary

Table 8-6 summarizes bit-map object functions.

Table 8-6. Bit-Map Functions	
Function Name	Description
GpiCreateBitmap	Creates a bit map that is compatible with a device associated with a presentation space.
GpiDrawBits	Copies bit-map image data from storage to a bit map that has been selected into a device context.
GpiImage	Draws an image primitive.
GpiLoadBitmap	Loads a bit map from a resource file or application EXE file.
GpiQueryBitmapBits	Transfers data from a bit map to application storage.
GpiQueryBitmapDimension	Determines the width and height of a bit map, in units of 0.1 mm.
GpiQueryBitmapHandle	Determines the handle of a bit map with a specific local ID.
GpiQueryBitmapInfoHeader	Determines information about a particular bit map, then transfers the information to the BITMAPINFOHEADER structure.
GpiQueryBitmapParameters	Determines information about a particular bit map, then transfers the information to the BITMAPINFOHEADER structure. Use GpiQueryBitmapInfoHeader whenever possible.
GpiQueryDeviceBitmapFormats	Determines the device bit map formats (number of planes and number of bits per pel) supported by the device, with the format that most closely matches the device returned first.
GpiQueryPel	Determines the color index or RGB value for a specific pel, whose location is specified in world coordinates.
GpiSetBitmap	Selects a bit map into a memory device context.
GpiSetBitmapBits	Transfers standard-format bit maps from a buffer into a bit map, associated with a memory device context.
GpiSetBitmapDimension	Defines the height and width of a bit map, in units of 0.1 mm.
GpiSetBitmapID	Assigns a local identifier to a bit map.
GpiSetPel	Draws a pel using the current line bundle color and mix attributes at a specified position in world coordinates.
WinDrawBitmap	Draws a bit map in a display window.

Table 8-7 summarizes the data structures used by the bit-map object functions.

Table 8-7. Bit-Map Structures

Structure Name	Description
BITMAPINFO2	Contains 20 fields that specify the attributes of the bit map, such as the bit-map width, height, number of color planes, bit count, RGB array structure, information header size, and so on.
BITMAPINFOHEADER2	Contains 19 fields that specify the bit-map header information, such as the bit map width, height, number of color planes, bit count, information header size, and so on. This record contains the same fields as BITMAPINFO2, with the exception of the last field, <code>argbColorY1</code>.
RGB2	A 4-byte structure containing 1 byte blue, green, and red color component values.

Chapter 9. Fonts

In typography, a *font* is a collection of characters that share a common height, line weight, and appearance. In presentation manager, fonts are an *lcid* identified resource, like bit maps, and are used in conjunction with the character string primitive data structures to define the appearance of displayed and printed text. The primary role of the PM programming interface with regard to fonts is to evaluate whether a particular font is appropriate for a given situation.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Character string primitives
- Coordinate spaces
- Transformations
- Color and mix attributes.

About Fonts

Font height is specified in printer's points, referred to in this volume as just *points*. A point equals approximately 1/72 of an inch. The line weight and appearance of a font are specified by the categories listed in Table 9-1.

Table 9-1. Examples of Line Weight and Font Appearance Categories		
Attribute	Category	Description
Line Weight	Normal	Characters drawn with a normal line weight.
	Bold	Characters drawn with a heavier line weight.
Appearance	Condensed	Characters drawn with ½ the average character width.
	Expanded	Characters drawn with twice the average character width.
	<i>Italic</i>	Characters drawn with a normal line weight and sloped to the right.

A *font family* is a collection of fonts that share common design characteristics such as stroke width and serif. *Stroke width* refers to the width of lines used to draw the characters and symbols of a font. A *serif* is a short cross-line drawn at the ends of the main strokes forming a character or symbol.

Text output is drawn with the characters and symbols of a font. The operating system stores fonts either in memory or on devices. Applications can access fonts through a device context associated with the current presentation space. When an application creates a presentation space, using `GpiCreatePS` or `WinGetPS`, the operating system assigns the presentation space a default font from one of the available fonts. An application can retrieve information about the default font using `GpiQueryFontMetrics`.

Image Font and Outline Font Implementation

The operating system can define and implement the characters of a font as either *images* or *outlines*.

Each of the characters in an image font (sometimes called a *raster font*) is created by alternating the ON and OFF settings of pels to produce the image. The image characters are stored as a bit map; sometimes image fonts are even referred to as *bit-map fonts*.

The characters in an image font must be drawn in a fixed size—you cannot rotate or scale them, for example—but usually they are of good quality; they are displayed faster than outline characters; and frequently, they look better at low resolutions.

The characters in an outline font, on the other hand, are drawn using a sequence of lines and arcs; then the characters are filled if so requested. The outline characters are stored as a collection of line, fillet, and spline functions. Outline fonts are *transformable* because the outline characters can be scaled, rotated, and sheared. Outline fonts can be implemented as vectors, drawn by a series of small, straight lines; if that is the case, they can be referred to correctly as *vector fonts*. Vector fonts can be drawn much faster than fonts requiring the inclusion of actual curves and arcs.

A major advantage of outline fonts is their device independence; unlike image fonts, whose appearance depends on the device's pel definition.

Figure 9-1 is an example of a character in both an image font and an outline font.

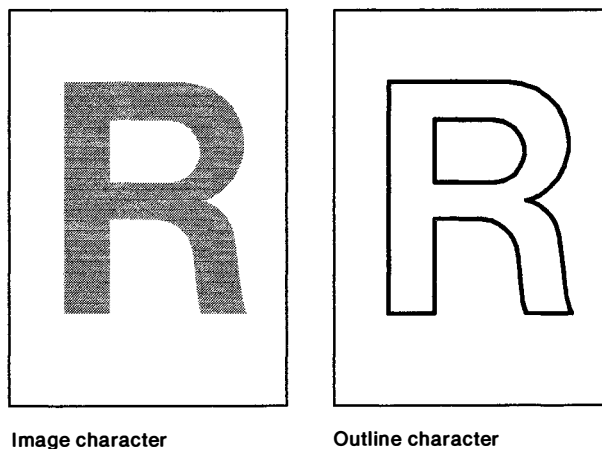


Figure 9-1. Image and Outline Characters. The image character is produced by manipulating pels. The outline character is produced by drawing a sequence of lines and arcs. (The resultant shape of an outline character can be filled if desired.)

Note: Most of the functions and data structures described in this chapter relate only to outline fonts.

PM-Supplied Fonts

The PM programming interface provides a number of both image and outline typographic-quality character fonts. On some devices, image fonts in small font sizes have a better appearance than their outline-font equivalents.

Type size is measured in *points*. There are approximately 72 points to an inch, so each character in a 24-point font, for example, is 1/3-inch high. Each incremental unit of a presentation page in PU_TWIPS format is 1/20 of a printer's point (1/1440 inch).

The following image fonts are available on all OS/2-supported display adapters:

<i>Table 9-2. Available Image Fonts</i>	
Font Family Name	Point Sizes Available
Tms Rmn	8, 10, 12, 14, 18, 24
Helv	8, 10, 12, 14, 18, 24
Courier (monospace)	8, 10, 12
System Monospace	10

In addition, a default system font is used in window components such as title bars and menus; it is a proportionally spaced Swiss font.

The outline fonts provided by the PM programming interface are Adobe® Type 1. The appearance and performance characteristics of these fonts are more flexible than for image fonts.

Table 9-3 lists the outline fonts available with the operating system and their equivalent fonts from earlier versions of OS/2.

<i>Table 9-3. Available Outline Fonts</i>		
Outline Font Family Name	OS/2 2.0 Fonts Available	Equivalent Name
Times New Roman®	Times New Roman	Tms Rmn
	Times New Roman Bold	Tms Rmn Bold
	Times New Roman Bold Italic	Tms Rmn Bold Italic
	Times New Roman Italic	Tms Rmn Italic
Helvetica®	Helvetica	Helv
	Helvetica Bold	Helv Bold
	Helvetica Bold Italic	Helv Bold Italic
	Helvetica Italic.	Helv Italic
Courier	Courier	Courier
	Courier Bold	Courier Bold
	Courier Bold Italic	Courier Bold Italic
	Courier Italic	Courier Italic
Symbol	Symbol	None
The outline font names provided with earlier versions of OS/2 are still supported, but the corresponding new fonts are obtained when the previous font names are selected.		

Availability of Additional Fonts

Over 600 Adobe Type 1 fonts are available from font vendors, and user systems could have a large number of these installed. Applications must be capable of functioning properly with whatever font the user selects.

An application can examine the font metrics to see what a font looks like or display different font characters from which the user can choose. When dealing with an interactive or user-driven application, the recommended method of choosing a font is to call `WinFontDlg`, which displays an example of the font in a dialog box.

Additional Adobe Type 1 fonts can be used just like other outline fonts. Notice, however, that some of those fonts are more *stylized*, that is, the fonts have a greater variation in the widths of different characters within the same font, which is provided by *Kerning*. Kerning is described and illustrated in “Kerning” on page 9-9.

Adobe Type 1 fonts, provided with and for a particular application, can be loaded with `GpiLoadFonts`, provided that the following rules are observed:

- The .AFM file of the font must be specified as the font file.
- The .PFB file must be in the same directory as the .AFM file.

Font Data Structures and Attributes

The attributes of fonts are contained in the FONTMETRICS data structure. The appearance of the actual text is influenced also by the attributes of the individual characters, which can be found in the CHARBUNDLE data structure. The character string primitives are described in Chapter 6, “Character String Primitives.”

An application can determine whether to use a particular font by examining its *font metrics*, which are the measurements that define the features of that font. The measurements are decided on by a font designer, whose most important criteria might be ensuring that the font is pleasing to the eye.

Unlike the attributes of a character string primitive, the individual font metrics attributes cannot be changed using specific GPI functions. Your application can determine the values of the current logical font attributes by calling `GpiQueryFontMetrics`, which accepts as input the amount of data to be returned as well as a pointer to the data area. Unlike other GPI calls, `GpiQueryFontMetrics` does not return the size necessary for all font metrics data. Querying with a `sizeof` operator, instead of calling the query twice, returns the size data.

The following table and text defines and explains the uses of the various font metric attributes.

Table 9-4 (Page 1 of 2). Font Metric Attributes	
Attribute	Description
Face name	The full typeface name of a font, such as <i>Courier Bold Italic</i> .
Family name	A broader equivalent of the face name (e.g. <i>Courier</i>).
Code page	The mapping between a set of codepoints and a set of graphic characters.
Character cell	Controls the height and width of outline font characters.

Table 9-4 (Page 2 of 2). Font Metric Attributes

Attribute	Description
Character cell ascent	Distance between the baseline to the top of the character cell.
Character cell descent	Distance between the baseline and the bottom of the character cell.
Maximum baseline extent	Maximum vertical extent of font characters in world coordinates.
Maximum ascender	Maximum distance a font character ascends above the baseline.
Maximum descender	Maximum distance a font character descends below the baseline.
Internal leading	Vertical distance equal to the difference between the maximum baseline extent and the em height.
Em height	Equivalent to the character cell height.
Em width	Equivalent to the character cell width.
X height	Height above the baseline of any lowercase character.
Lowercase ascent	Maximum distance of a lowercase character above the baseline.
Lowercase descent	Maximum distance of a lowercase character below the baseline.
External leading	Maximum vertical font spacing that can be added without adversely affecting the appearance of the text.
Average character width	Average width of font characters in world coordinates
Character slope	Initial slope of a character with respect to a vertical line.
Inline direction	Character angle measured with respect to a horizontal line.
Weight class	Specifies the thickness of each stroke of a font character.
First, last, default, and break characters	See "Glyphs, Code Pages, and Code Points" on page 9-8.

The *face name* attribute is the typeface name by which a particular font design is known. Two examples are Times New Roman Bold and Times New Roman Italic. Roman is the *family name* of these fonts. You always should provide a face name because most outline fonts are known by it.

The *code page* to be associated with the font is identified by this metric. Some fonts are specific to a particular code page and should be used with only that code page. Other fonts are universal and can be used with any supported code page. Code pages are described in "Glyphs, Code Pages, and Code Points" on page 9-8.

Every character in a font is drawn within a rectangular region called a *character cell*. The character-cell height is the distance from the bottom of the character cell to its top. The width of the character cell is the distance from one side to the other.



A font's *Em height* is a measure of its visual height. This measurement was given its name because, historically, the height of an uppercase letter M usually was equal to the average height of all uppercase characters in the font. However, this is no longer the case.

The value of Em height represents the font point size in world coordinates and is the same as the character cell height. For an outline font, this can be set by the character cell height attribute. The value of *Em Width* is the equivalent horizontal dimension and is the same as the character cell width. For an outline font this also can be set by the character cell width attribute. These are, in fact, the dimensions of the *em square* (a typographic term) and, like point size, cannot be defined in terms of any measurable characteristic of a visible character of the font.

The average distance from the baseline to the top of any lowercase character is called a font's *x height*. This measurement was given its name because the height of a lowercase x usually is equal to the average height of all lowercase characters in the font.

The *maximum ascender* is the height of the tallest character in a font. The *maximum descender* is the depth (below the baseline) of the lowest character in a font.

The *lowercase ascent* is the height of the tallest lowercase character in a font. The *lowercase descent* is the depth (below the baseline) of the lowest lowercase character in a font.

Many fonts reserve part of the space in the top of each character cell for accent marks. This reserved space is called *internal leading*. Some fonts require a certain amount of space to be left between rows of text. This space, called *external leading*, is not included in the character-cell height or ascent measurements. The external leading is the recommended space to leave between rows of text to preserve its pleasing appearance.

The *average character width* is stored in the **IAveCharWidth** field in the FONTMETRICS structure. The average character width is a measure of character width based on the normal frequency of lowercase letter usage in the English language. You can use this value to approximate the average width of a character in string length calculations.

The *maximum baseline extent* for a font is the sum of the maximum ascender and the maximum descender. Maximum baseline extent is not equal to cell height for outline fonts, but is for image fonts.

The *character slope* is an angle measured clockwise with respect to a typical vertical line. The slope of a normal font is 0; the slope of an italic font is other than 0. The character slope in the FONTMETRICS data structure can be thought of as the initial slope, since the slope can be changed in the CHARBUNDLE data structure.

The *inline direction* is an angle measured clockwise with respect to a horizontal line. The inline direction for a Swiss, Helvetica, or Roman font is normally 0—exactly horizontal. The inline direction for a Hebrew font is normally 180, because Hebrew text is oriented right to left.

The *character-rotation angle* is an angle measured counterclockwise with respect to a horizontal line. The baselines of characters are aligned with the vector drawn at the angle of rotation.

The weight class specifies the thickness of each stroke that forms part of each character in a font.

The first, last, default, and break character all are associated with glyphs for UGL fonts and code pages and are described in “Glyphs, Code Pages, and Code Points.”

A *superscript* is a character drawn immediately above and to the right of a normal character in a string of text. Superscripts are identified by a width and height and by vertical and horizontal offsets.

A *subscript* is a character drawn immediately below and to the right of a normal character in a string of text. Subscripts are identified by a width and height and by vertical and horizontal offsets.

Kerning pairs are described in “Kerning” on page 9-9.

Glyphs, Code Pages, and Code Points

The image or picture that you associate with each character or symbol in a font is called a *glyph*. The mapping between a set of glyphs and their code points is called a *code page*.

For a single byte character set (SBCS), each code page contains up to 256 code points. Normally these are 8-bit integers in the range 0 through 255, with 1 code point identifying 1 glyph in that code page. The code pages can be either ASCII or EBCDIC. Because a code point repeats on a new code page, you need both the code page and code point to uniquely identify a glyph.

There usually is 1 code page per font; however, double byte character sets (DBCS) are sometimes used for Asian languages with large character sets. Fonts are a set of glyph definitions and a default code page mapping. The same set of font glyph definitions is remapped for different code pages.

Each font contains 4 special glyph points:

- First glyph
- Last glyph
- Default glyph
- Break glyph.

First and last glyph points are of more interest to the font designers than application programmers because they apply to the set of font glyph definitions rather than a particular code page. The last character is the maximum code point of the font that has a glyph associated with it and may be bigger than 256.

The default glyph appears in text when an application specifies a glyph point that does not exist in the font.

The break glyph usually is the space character and often has the same code point as the default character.

Actually a font can have more or less than 256 definitions, but any particular code page may have less than 256 but no more. There may be a translation between the code point in the code page and the code point (between the first and last glyphs) that references the glyph in the font.

The operating system assigns unique identifiers to each of its code pages. Common code pages are 437—the United States code page, and 850—the multilingual code page. The default code page is 850.

An application can determine the current code page by calling `GpiQueryCp`, or it can assign a new code page using `GpiSetCp`. If you default the code page in the `FATTRS` structure when calling `GpiCreateLogFont`, you get the current code page as specified by `GpiSetCp` (or returned by `GpiQueryCp`). You can specify any one of the code page identifiers returned to you from the `WinQueryCpList` function.

When using a font other than the system font, you specify the required code page in the `FATTRS` structure of `GpiCreateLogFont`. You can determine the code page for that font (if it is the current logical font) by using `GpiQueryFontMetrics`. The available code pages are described in the *Presentation Manager Programming Reference*.

Proportional and Monospace Fonts

When text is drawn, the operating system aligns each character by positioning its character cell next to that of the previous character.

A monospace font provides the same amount of space for each character, whatever its shape. These fonts also are called *fixed fonts*. Courier and System Monospaced are monospace fonts. Monospace fonts, in fact, are essential for some purposes, for example program listings, where vertical alignment is important.

A proportionally-spaced font is one in which some characters are allotted less space than others, so that each character occupies the proportion of space that is correct for its shape. For example, a lowercase letter *l* does not need the same space as a lowercase letter *m*. Many graphics fonts supplied by the PM programming interface, including the system font, support proportional spacing.

The font metrics value **maxcharinc** specifies the width of the widest character in the font, and the value **avecharwidth** specifies an average width of the characters in the font.

You can retrieve information about the widths of the characters in the current font by calling `GpiQueryWidthTable`. You provide the code point of the first character you are interested in and the number of characters for which you want width-table information. Refer to the *Presentation Manager Programming Reference* for information on how character-width information is computed.

Kerning

Kerning, like proportional spacing, is a type of letter spacing. In a kerned font, some characters are allowed to overhang others and, therefore, occupy an area that is less than each character's increment for that font. Usually this feature is restricted to pairs of characters whose appearance might benefit from greater proximity. The 5 character pairs that most commonly benefit from kerning are *Yo*, *We*, *To*, *Tr*, and *Ta*.

Kerning is not available in the system-provided image fonts. The Helvetica and Times New Roman outline fonts provided with the system do include kerning information. You can specify kerning as a requirement on the `GpiCreateLogFont`. Selecting a kerned font lets a kerning take place whenever it is defined.

The *a-space* is the area of the character cell before the character; the *b-space* is the area of the character cell occupied by the character; and the *c-space* is the area of the character cell after the character. The *a-space* and *c-space* are shown in Figure 9-3.

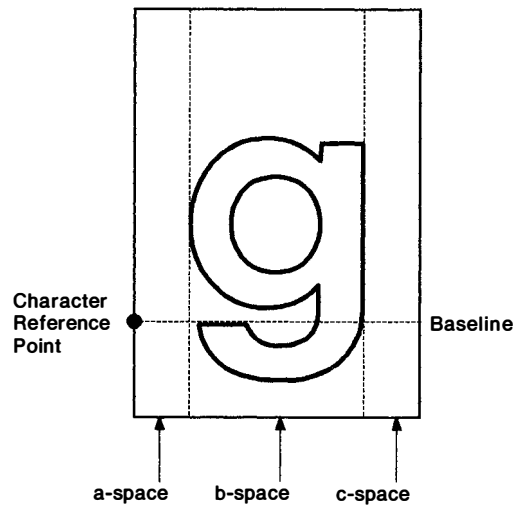


Figure 9-3. *a-Space, b-Space, and c-Space*

The best way to implement kerning is to use a kerning-pair table. The kerning table will exist only for certain pairs of characters for which adjustments are desirable.

Each of the entries in a kerning-pair table contains the code points of a pair of kerned characters and an adjustment that should be applied to the character width of the first character of the pair. A negative value means that the space is decreased, and a positive value means it is increased, whenever the 2 characters in a kerning pair are produced together. A kerning-pair table might be provided with a purchased font. If so, it will be recorded in the font header.

If kerning is specified on a character-pair basis (that is, if the font has a kerning-pair table), the font has a specific number of kerned pairs. `GpiQueryKerningPairs` returns kerning-pair information from the current logical font. For each pair of kerned characters, you are given the characters themselves and an adjustment in world coordinates that must be applied to the character width of the first character of the pair.

When you specify negative *a-* and *c-space* values, the values affect *any* character paired with the kerned character. Because this form of kerning cannot be applied selectively, you have to choose the kerned characters carefully. For example, an italic letter *f* (*f*) is a good candidate for this sort of kerning, because it can overhang most of the characters that precede and follow it.

An application can adjust the amount of space between all characters in a font with `GpiSetCharExtra`. Any extra space is added to font kerning values. An application also can adjust the width of a font's break character (space) with `GpiSetCharBreakExtra`. These functions are described in Chapter 6, "Character String Primitives."

Kerning is not available on all devices. On devices that support kerning, kerning is enabled by default. When kerning is not supported on a device, kerning support is switched off by default. When kerning support is switched off, the kerning

information supplied with a font is ignored. To determine whether a device supports kerning, use `DevQueryCaps`.

The kerning information can be implemented by your application when a character string primitive is written to an output device; this is called *rendering* the text. The functions `GpiCharStringPos` and `GpiCharStringPosAt`, both described in Chapter 6, "Character String Primitives," permit specification of the starting position for each character. The entire character string can be searched for character pairs that also are kerning pairs for the kerning information to be applied.

FATTRS Data Structure

The FATTRS structure is used to identify the characteristics of a requested font. Certain attributes of a font that govern many individual character string primitives are contained in FATTRS, as follows:

- Record length
- Selection indicators
- Match value
- Face name
- Registry identifier
- Code page
- Maximum baseline extent
- Average character width
- Type indicator
- Font use indicator.

Record Length

Record length determines the length of the FATTRS structure. An application normally specifies this with the `sizeof` operator.

Selection Indicators

There are 5 selection indicators:

Indicator	If set ON, selects...
FATTR_SEL_ITALIC	A font with an italic appearance. This is intended for use with an image font and results in a simulation of an italic font.
FATTR_SEL_BOLD	A boldface version of the font. This is intended for use with an image font and results in a simulation of a boldface font.
FATTR_SEL_OUTLINE	Hollow characters when using an outline font. This only can be used with outline fonts. This indicator is less important than the others because improved performance is now available (from Adobe Type Manager) on filled outline fonts than was originally available when this option (FATTR_SEL_OUTLINE) was introduced.
FATTR_SEL_STRIKEOUT	A font in which the characters are struck through. This option also might be selected on the character string primitive.
FATTR_SEL_UNDERSCORE	A font in which the characters are underscored. This option also might be selected on the character string primitive.

Any combination of these indicators can be set. When set, they cause a *simulation* of the selected feature. For example, if you require a bold, italicized version of a PM image font, PM simulates that feature because bold, italic image fonts are not supplied.

Match Value

The *match value* is a number that identifies a physical font; it is allocated to the font when the font is loaded. If the match value is less than 0, the font is a *device font*; and, if the match value is greater than 0, the font is a *generic font*.

A device font is exclusive to the device with which it is associated, and its match number is guaranteed to be unique only within the device driver. If an application prints on a different printer, the application cannot guarantee that the font will be identical just by specifying the match value. A generic font is available on more than one device. The PM-supplied fonts, for example, are generic fonts. Printer fonts always are device fonts, but you also can and should use generic fonts with printers.

You can choose whether or not to use the match value to specify the font when calling GpiCreateLogFont. Specifying the match value is easier but less flexible than the alternative, which is to specify the font name and outline characteristics. For an image font, the alternative is to specify the maximum baseline extent and average character width of the required font.

Face Name

The *face name* value in the FATTRS data structure is identical to the face name of the FONTMETRICS data structure when describing a specific font.

Registry Identifier

The *registry identifier* is a unique number by which the font is registered with IBM. The registry identifier of a particular font can be retrieved by calling GpiQueryFonts. If you do not have a registry number, set this value to 0.

Code Page

The *code page* is the identifier of the code page to be associated with the font. You can default the code page in GpiCreateLogFont by specifying 0. Like the face name, the code page in the FATTRS data structure is identical to the code page in the FONTMETRICS data structure when describing a specific font.

Maximum Baseline Extent

The *maximum baseline extent* is the vertical space occupied by characters in the font. If you are setting the font-use indicator FATTR_FONTUSE_OUTLINE, described on page 9-13, you should set the maximum baseline extent to 0. Outline fonts take an equivalent value from the character cell attribute that is current when text is written to an output device.

The maximum baseline extent value is required to select an image font and must be specified in world coordinates. For image fonts, this is the vertical height in world coordinates of character images in the font. This field must be specified when requesting an image font with GpiCreateLogFont.

The maximum baseline extent measurement is shown in Figure 9-2 on page 9-6. The maximum baseline extent in the FATTRS data structure is used for programming, unlike the maximum baseline extent in the FONTMETRICS data structure, which is only measurement as recommended by the font's designer.

Average Character Width

The *average character width* of a font is the average character increment in world coordinates. The *character increment* is the sum of the a-space, b-space, and c-space of a character, which are shown in Figure 9-3 on page 9-10. If you are setting the font-use indicator `FATTR_FONTUSE_OUTLINE`, you can set the average character width to 0 because the outline fonts take an equivalent value from the character cell attribute that is current when text is written to an output device.

This field should be specified when requesting an image font using `GpiCreateLogFont`; it must be specified in world coordinates.

Type Indicator

PM provides a *type indicator* to allow the further specification of font most appropriate to your application. The type indicators found in the `FATTR` data structure are not interchangeable with the type indicators in the `FONTMETRICS` data structure.

The most commonly used type indicator, `FATTR_TYPE_KERNING`, causes kerning to take effect for those fonts that contain kerning-pair information. `FATTR_TYPE_KERNING` is supported only for PostScript™ devices.

There are three other type indicators provided:

Indicator	When...
<code>FATTR_TYPE_ANTIALIASED</code>	An anti-aliased font is required
<code>FATTR_TYPE_MBCS</code>	A mixed single- or double-byte code page is required
<code>FATTR_TYPE_DBCS</code>	A double-byte code page is required.

Refer to the *Presentation Manager Programming Reference* for details about these type indicators.

Font-Use Indicators

The font-use indicators tell the PM programming interface what you intend to do with the font, and determine whether you get an image font or an outline font. There are three font-use indicators:

Indicator	If set ON...
<code>FATTR_FONTUSE_OUTLINE</code>	Forces selection of an outline font. This value must be set if you intend to use graphics characters in a path definition.
<code>FATTR_FONTUSE_TRANSFORMABLE</code>	Allows the scaling, rotating, or shearing of font characters. Set this value ON if the current character mode is <code>CM_MODE3</code> .
<code>FATTR_FONTUSE_NOMIX</code>	Allows the PM programming interface to ignore current mix-mode values. Set this value ON if the graphics text is not going to be written over or underneath other graphics, because it improves the performance of text drawing.

`FATTR_FONTUSE_TRANSFORMABLE` and `FATTR_FONTUSE_OUTLINE` normally are both specified to select an outline font. Specify neither if you want to select an image font.

Public and Private Fonts

The fonts supplied by PM are loaded automatically when the system is started and are available to any application at any time. They are said to be *public fonts*. Any font that you define locally has to be loaded before you can use it. If you load a locally defined font using the **Install** option of the Workplace Font Palette, that font is available to any application in the system, and, therefore is a public font. Some devices provide their own fonts. All device fonts are public fonts.

Any font loaded using GpiLoadFonts from within an application is a *private font*. A private font is available only to the application that loads it. The file in which you store a locally created font definition can contain more than one font definition. GpiLoadFonts loads a named font file and, therefore, loads all fonts defined in that file. GpiQueryFontFileDescriptions provides the family name and the face name of each font in a locally defined font file; use this function to determine if a particular file contains the font you want to use before you load it.

To unload a previously loaded private font file, call GpiUnloadFonts. You cannot unload public fonts with this function.

You cannot use private fonts in a document that you want to print using the spooler. If you want to print the document from a printer server, using the PM_Q_STD format, all the fonts used in the document must be installed as public fonts on the server.

You can use either public or private fonts if you specify the data format PM_Q_RAW or if you want to print without the spooler.

Drawing a Character String Primitive

To draw a string of one or more graphics characters from the current font, call GpiCharString. The string starts at the current position and, when it has been drawn, the new current position is the point at which the next character would be, had there been one.

To draw the string starting at a position other than the current position, call GpiCharStringAt. This is equivalent to calling GpiSetCurrentPosition or GpiMove, followed by GpiCharString. You supply the text of the character string as a parameter to the appropriate GPI function.

Note: You also can call WinDrawText, which draws text to the screen a line at a time. WinDrawText differs from the GpiCharStringxxx functions in that WinDrawText ignores the DM_RETAIN drawing mode and can only be used to draw to a window device context.

Figure 9-4 shows some examples of a character string primitive.

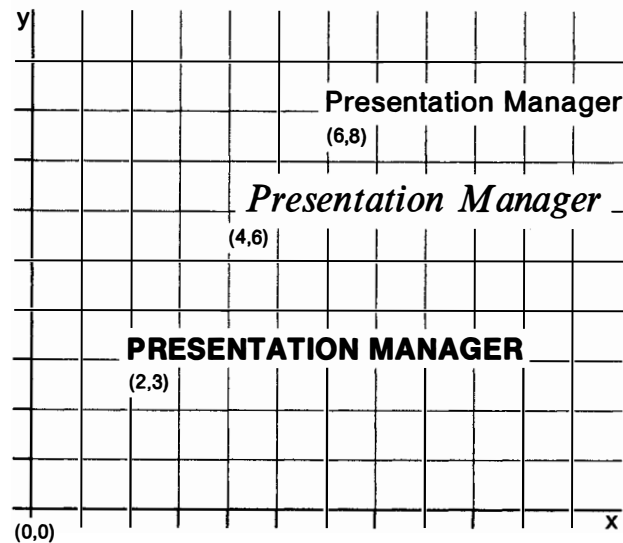


Figure 9-4. The Character String Primitive. This example shows the string Presentation Manager drawn from different logical fonts.

Font Files and Dynamic-Link Libraries

You can use the Font Editor to alter and customize image-font files. Files created with Font Editor have a .FNT extension. After you create a custom font, put it into a dynamic-link library (DLL) that the application can load. Once the application loads this library, it can use any of the custom fonts the library contains.

DLLs that contain fonts are unique; they contain data segments and empty (or useless) code segments. They are unique because they contain no executable code, unlike normal DLLs. Also you are able to install the font file as a public font if desired; GpiLoadFonts will make it available only as a private font. The operating system naming convention for a DLL that contains font information end with a .FON extension.

To create a DDL, you must use an assembler, a linker, and the operating system Resource Compiler. For this example, assume that your custom font file is named NEWFONT.FNT.

After creating your custom font file, you need to create a special assembler language file with your editor. Call this file MYFONT.ASM, for example. Figure 9-5 shows an example of the source code for this file.

```
code segment word      ;Makes dummy code segment aligned on word boundary
db "empty_segment"    ;Initializes a string in dummy segment
code ends              ;Dummy segment ends here
end                    ;Terminates source file
```

Figure 9-5. Creating a Custom Font

After you have created MYFONT.ASM, assemble it by using the following command:

```
masm myfont
```

After assembling the MYFONT.ASM file, create a module-definition file. Call this file MYFONT.DEF, for example. It should contain the statements in Figure 9-6.

```
LIBRARY myfont  
SEGMENTS CODE MOVEABLE
```

Figure 9-6. Module Definition File for Custom Font File

The first statement tells the linker that you are creating a library with the module name, MYFONT. The second statement tells the linker that the segments in this library are movable code segments.

After creating MYFONT.DEF, start the linker with the command in Figure 9-7.

```
link myfont,...,myfont.def
```

Figure 9-7. Linking a Custom Font File

This command creates an empty executable file called MYFONT.EXE.

After creating the empty executable file, which is the template for a DLL, create a resource file and call it MYFONT.RC. For example, if your font file is called NEWFONT.FNT, you would place the statement in Figure 9-8 in MYFONT.RC.

```
FONT 200 NEWFONT.FNT
```

Figure 9-8. Resource for Custom Font File

This statement assigns the identifier, 200, to the font resource NEWFONT.FNT.

Finally, use the Resource Compiler to add the font file (NEWFONT.FNT) to the empty DLL as in Figure 9-9.

```
rc myfont.rc
```

Figure 9-9. Creating the Resource File for a Custom Font File

The executable file, MYFONT.EXE, now contains your custom fonts. After you have renamed this file, MYFONT.FON, you can load the fonts into your application using GpiLoadFonts and pass it a pointer to the path and library name as the second argument—for example, c:\os2\dll\myfont.fon.

Using Fonts

You can use the font functions to:

- Determine the characteristics of available fonts
- Select a font for text output
- Delete or unload a font when finished
- Create a font
- Create marker sets and pattern sets
- Use a bit map as a fill pattern
- Use a clip path as a fill pattern.

Selecting a Font

Every presentation space has a *current font*, and graphics characters must be drawn in that font. By default, the current font is the system font. Before an application can use any font other than the system font, it has to identify the other font as the new current font.

There are 2 distinct ways to identify the other font:

- Call `GpiQueryFonts` to request information about the available physical fonts, and then explicitly select one of those fonts by supplying its identifier as input to `GpiCreateLogFont`.

This process has 11 steps and is described in "Explicit Font Selection." This method of selecting a font is most useful for selecting image fonts.

If you must have an image font, you are urged to use this "explicit selection" method, because you need to specify a maximum baseline extent and an average character width in the `FATTRS` structure. These values can be determined only by calling `GpiQueryFonts`. If you specify values that are not valid, you are likely to be given an outline font.

- Use `GpiCreateLogFont` to specify the features you require in a font, and permit the PM programming interface to make an appropriate font available to you.

This process has 2 steps and is detailed in "Closest-Matching Font Selection" on page 9-22.

Explicit Font Selection

An application can select either a public or a private font with `GpiCreateLogFont`. A public font is available to all applications. A private font is loaded by an application for its exclusive use.

Use the PM Control Panel to load a public font. Four DLLs contain the Times Roman, Helvetica, Courier, and System Monospaced image fonts. The names of these libraries are:

- `TIMES.FON`
- `HELV.FON`
- `COURIER.FON`
- `SYSMONO.FON`.

Unlike most DLLs, font libraries typically use the file name extension `.FON`. If the user loads all four libraries, a total of 76 public fonts are available. An application can use both outline and image formats of these fonts. Characters in the image format are available in sizes ranging from 8 to 24 points. Characters in the outline format can be any size.

Call `GpiLoadFont` to load a private font. Pass the function the path and name of the DLL that contains the font. After the application loads the DLL of fonts, it can determine the characteristics of the fonts in that library by calling `GpiQueryFonts`.

To select a public font from all of the available public fonts, do the following:

1. Call `GpiQueryFonts`, passing the `QF_PUBLIC` flag, a NULL pointer to the font's face name, and a count of 0 to determine the number of available public fonts, as shown in Figure 9-10.

```
#define INCL_GPILCIDS
#include <os2.h>
void fncFONT06(void){

    FONTMETRICS fm, afm[80];
    LONG cFonts = 0, cPublicFonts;
    HPS hps;

    cPublicFonts = GpiQueryFonts(hps,      /* Queries all public fonts */
    QF_PUBLIC,
    (PSZ) NULL,
    &cFonts,
    sizeof(fm),
    (PFONTMETRICS) NULL);
} /* fncFONT06 */
```

Figure 9-10. Determining the Number of Public Fonts Loaded

Note: To load and use a private font, follow the same procedure, but specify `QF_PRIVATE`, instead of `QF_PUBLIC`, in the calls to `GpiQueryFonts`.

2. Copy the value returned by `GpiQueryFonts` into a LONG integer variable.
3. Allocate memory for the FONTMETRICS data structures.
4. Call `GpiQueryFonts` again, passing the `QF_PUBLIC` flag, a NULL pointer to the font's face name, the count returned by the previous call, and the address of an array of FONTMETRICS structures.

`GpiQueryFonts` returns the font metrics of every font available the array of FONTMETRICS structures. The font metrics define in detail the physical characteristics of a font and are described in the *Presentation Manager Programming Reference*.

Because the font metrics are so detailed, the amount of information returned to you from `GpiQueryFonts` can be extensive. You can restrict the amount of information returned by this function by:

- Specifying an absolute number of fonts about which you require information
- Asking for information about fonts of a specific face name only
- Limiting the length of the font-metrics buffer.

If you request information for more fonts than are available on the system, `GpiQueryFonts` returns all the information about the available fonts and a value indicating how many fonts it has returned.

If you request information for fewer fonts than are available on the system (that match the specified face name, and so forth), `GpiQueryFonts` returns a value indicating the number of fonts that it was unable to return.

Figure 9-11 is an example of this technique of selecting a font.

```
#define INCL_GPILCIDS
#include <os2.h>

PFONTMETRICS fncFONT07 (HPS hps, PLONG pcPublicFonts) {
    PFONTMETRICS afm = NULL;
    LONG          fonts = 1000;

    *pcPublicFonts = GpiQueryFonts(hps,
                                   QF_PUBLIC,
                                   (PSZ) NULL,
                                   &fonts,
                                   sizeof(*afm),
                                   (PFONTMETRICS) NULL);

    if (lDosAllocMem(&afm,
                    sizeof(*afm)*(*pcPublicFonts),
                    PAG_COMMIT | PAGWRITE )) {
        GpiQueryFonts(hps,
                      QF_PUBLIC,
                      (PSZ) NULL,
                      pcPublicFonts,
                      sizeof(*afm),
                      afm);
    } /* endif */
    return afm;
}
```

Figure 9-11. Selecting a Font

5. Examine the array of FONTMETRICS structures.

- a. For an outline font, examine the face name and other attributes of the font your application requires.
- b. For an image font, examine the device resolution fields. These fields are the ones that determine if a particular font and its metrics match the device with which you wish to use it.

Note: If your application is interactive, organize the face names and other information relevant to those fonts (such as point sizes) in a menu. When the operator has selected a font, supply the relevant information from the font metrics of that font in the FATTRS structure of GpiCreateLogFont.

6. After determining which font to use, your application must fill in the fields of the FATTRS structure. Some of the data for these fields will be copied (explicitly) from the FONTMETRICS structure. Then call GpiCreateLogFont.

If you do not supply a face name, the default font is used. If you supply a face name, and the presentation driver cannot find a matching font, a default font is selected.

The FATTRS structure describes a logical font. An application can have up to 254 logical fonts defined at any one time in a single presentation space.

Note: A *logical font* is a list of font attributes, whereas a *physical font* is the bit-map or vector information that the system uses to draw characters.

7. The fonts supplied by the PM programming interface, the fonts you load using the Control Panel, and any fonts you load with GpiLoadFonts are all *physical fonts*.

Copying the entries in the FATTRS structure ensures that, if a particular font is unavailable, an attempt is made to find the most suitable substitute. Without the FATTRS information, PM is less likely to locate a suitable font.

8. Initialize a local identifier (lcid) for the new font.
9. Call GpiCreateLogFont and passing as input parameters:
 - The lcid for the font
 - The address of an 8-character array containing the name you want to give to the logical font, (or, if you do not want to specify a name, the value NULL)
 - The address of the FATTRS structure.
10. Examine the return value from GpiCreateLogFont. The value will be 2 if the function is successful; 1, if the default font was used.
11. Pass the lcid to GpiSetCharSet, assigning the font to your application's presentation space. Then the application can use it for text output. (This step identifies a logical font as the current font.)

Figure 9-12 shows how to select an image font at least 14 device coordinate units high.

```

#define INCL_GPILCIDS
#include <os2.h>

LONG fncFONT08(HPS hps, LONG lcid, LONG xres, LONG yres) {
    FATTRS      fat;
    PFONTMETRICS afm;
    LONG         cFonts;
    LONG         i;
    LONG         rc = 0;

    afm = fncFONT07(hps, &cFonts);
    if (afm) {
        for (i=0; i<cFonts; i++) {
            if (!(afm[i].fsDefn & FM_DEFN_OUTLINE)
                afm[i].sXDeviceRes == xres &&
                afm[i].sYDeviceRes == yres &&
                afm[i].lMaxBaselineExt >= 14) {
                fat.usRecordLength = sizeof(fat);
                fat.fsSelection = 0;
                fat.lMatch = 0;
                strcpy(fat.szFacename, afm[i].szFacename);
                fat.idRegistry = afm[i].idRegistry;
                fat.usCodePage = 0;
                fat.lMaxBaselineExt = afm[i].lMaxBaselineExt;
                fat.lAveCharWidth = afm[i].lAveCharWidth;
                fat.fsType = 0;
                fat.fsFontUse = 0;

                rc = GpiCreateLogFont(hps, (PSTR8) NULL, lcid, &fat);
                if (rc) {
                    GpiSetCharSet(hps, lcid);
                } /* endif */
            } /* endfor */
        } /* endif */

        return rc;
    }
}

```

Figure 9-12. Selecting an Image Font. *GpiQueryFonts* returns device coordinates for image fonts. For outline fonts, it returns notional coordinates. Notional coordinates are the coordinate in which the font was defined. Usually outline fonts are defined over a 1000-by-1000 matrix, with the unit of the matrix a 1/1000 of the em height.

Reassociating the Presentation Space

The match value of any device font is guaranteed to be unique only for the current device. If you associate the presentation space with a different device context, any logical fonts that also are device fonts are no longer available. On the new device, the match number could identify a different device font (which may not be suitable) or no font at all. Therefore, you should redefine a logical font and select it as the current font if you reassociate the presentation space when the current font is a device font. Otherwise, the default font is used.

If you do not intend to use the font on the new device, however, you need not redefine it. You will be able to continue using the font if you reassociate the presentation space with the original device context.

Any generic fonts can continue to be used after the presentation space is reassociated if they are suitable for the new device. An image font, for example, can be used only on devices that are all-points addressable.

Font Resolution

Device fonts, particularly printer fonts, usually are designed for a specific resolution, namely the resolution of device on which they are to be used. By contrast, generic outline fonts are not device-specific. Generic image fonts are designed for a particular resolution value and are therefore really applicable only to devices with a matching resolution value.

The horizontal and vertical font resolution values for the device are returned by `DevQueryCaps`. The desired horizontal and vertical device resolution values in the font metrics of the font concerned returned by `GpiQueryFonts`. Compare these the device and the desired resolution values only for an image font. If they are identical then this image font is designed for use with this device (for example, the point size in the font metrics will be accurate for this device). To select a font, do the following:

1. Query the available fonts using `GpiQueryFonts`.
2. If you want an outline font, search the metrics of fonts returned by `GpiQueryFonts` for a font that is an outline font and that has the required face name.
3. If you want an image font, search the metrics of the fonts returned by `GpiQueryFonts` for a font that is an image font and that has the following:
 - Required face name
 - Required point size
 - Horizontal device resolution font metric that matches the horizontal resolution for the device
 - Vertical device resolution font metric that matches the vertical resolution for the device.
4. As an alternative to the above for an image font, search the metrics of the fonts returned by `GpiQueryFonts` for a font that is an image font and that has the following:
 - Required face name
 - Required em height in world coordinates.

Before you define a logical font, determine the resolution of the current device by using `DevQueryCaps` with the `CAPS_VERTICAL_FONT_RES` and `CAPS_HORIZONTAL_FONT_RES` parameters.

Compare the values returned from this function with the **`xDeviceRes`** and **`yDeviceRes`** values in the font metrics of the available fonts. These 2 values contain the resolution of the device for which the font was designed. All measurements are in pels per inch. By comparing the font resolution with the device resolution, you can identify the image font best suited to the current device.

Closest-Matching Font Selection

The alternative to selecting a specific font is to set the match value to 0. To have the PM programming interface select the closest-matching font available, do the following:

1. Define a logical font with `GpiCreateLogFont`. This function establishes a link between the calling application and an appropriate physical font. `GpiCreateLogFont` accepts as input a number of options, including the *font attributes*, which describe the physical features and capabilities of the font.

2. Complete the FATTRS structure of GpiCreateLogFont. The FATTRS structure describes a logical font, which the system uses to find the closest matching physical font. Depending on whether you are selecting a font using a match number, fill in the FATTRS structure, observing the rules listed in Table 9-5.

Table 9-5. Filling in the FATTRS Structure

If using a match number	If not using a match number
• Set the match number to the required FONTMETRICS value. The FONTMETRICS value is the value in the FONTMETRICS structure for the required font returned by GpiQueryFonts. • Set the maximum baseline extent to zero. • Set the average character width to zero. Note: A negative IMatch is only unique for a device. A positive IMatch is only unique on a particular system at a particular time.	• Set the match number to zero. • If your font is an image font: – Set the maximum baseline extent to the required FONTMETRICS value. – Set the average character width to the required FONTMETRICS value.

Also observe the rules concerning the font-use indicator that are listed in Table 9-6.

Table 9-6. Font-Use Indicator Considerations

If you have <i>not</i> set the FATTR_FONTUSE_OUTLINE indicator:	If you have set the FATTR_FONTUSE_OUTLINE indicator:
• PM looks for an image font that has the required selection indicators and whose maximum baseline extent and average character width match those you have specified. • If no suitable image font is found, PM looks for an outline font with the required face name and selection indicators. • If no suitable outline font is found, the default font is made available to you.	• PM looks for a suitable outline font whose selection indicators match those you have specified. • If no suitable outline font is found, the default font is used. No attempt is made to substitute an image font.

Outline fonts are affected by the current character attributes (for example, character box, shear, and angle). Because an outline font might be made available to you, even when you requested an image font, call GpiQueryFontMetrics to determine whether the font is an image font or an outline font. GpiQueryFontMetrics returns the metrics of the current logical font in world coordinates.

The settings of the definition indicator, **field defn** in the font metrics, tell you whether the font is either an image or outline font, or a device or generic font. If the font is an outline font, specify values for the character attributes before using the font. Outline fonts are unaffected by the maximum baseline extent and average character width values specified in `GpiCreateLogFont`.

If the default font provided is an image font, and you also have specified `FATTR_FONTUSE_TRANSFORMABLE`, any attempt to draw graphics characters in `CM_MODE3` raises an error condition. This is an exception to the general rule that a `FATTR_FONTUSE_TRANSFORMABLE` font must be used in `CM_MODE3` only.

Selecting the New Current Font

In addition to completing the `FATTRS` structure, you must supply as input to `GpiCreateLogFont` a unique `lcid` for the font. The identifier is a number in the range of 1 through 254. Because you can have more than one logical font definition in a presentation space, you must select the current font by supplying the font's `lcid` as input to `GpiSetCharSet`.

You do not have to call `GpiSetCharSet` to use the default font. When you want to reset the current font to its default value, however, call `GpiSetCharSet` with the value `LCID_DEFAULT`. The current font, like other modal attributes, reverts automatically to its default value when specific GPI functions are called.

Deleting Logical Fonts and Unloading Physical Fonts

A logical font can be used only by the application that defines it. The logical font definition is stored in the presentation space and is saved on the LIFO stack with its `lcid` when `GpiSavePS` is called. Unlike a physical font definition, a logical font definition is temporary and is lost when the presentation space is deleted.

To specifically delete a logical font, call `GpiDeleteSetId`, which deletes the logical font, and makes its `lcid` available for reuse. Sometimes this is necessary because there are only 255 `lcids` available. The logical font must not be the current font when you call `GpiDeleteSetId`.

Logical fonts are deleted to release the `lcid` and free memory. Unload private fonts using `GpiUnloadFonts` to free memory and to make them no longer available. Uninstall public fonts using the Control Panel to free memory and to make them no longer available.

You must end the association between physical and logical fonts *before* calling `GpiUnloadFonts`.

If you call `GpiUnloadFonts` to unload a private font file, you must end the association between the physical fonts in that file and the logical fonts defined by the application. Do the following:

- Call `GpiSetCharSet` to select a current font (such as the default font) that is not linked to the physical fonts you are unloading
- Call `GpiDeleteSetId` to delete each logical font that refers to any of the physical fonts in the font resource-file. If you fail to do this, an error is returned to your application from `GpiUnloadFonts`.

Creating Fonts

You cannot edit the definitions of any of the PM-provided fonts, although you can create and edit image-font files using the Font Editor. A font file contains a header, that describes the font in general terms and a section containing bit maps of the characters themselves. The font definition is stored in a resource file. A full description of the font-file format is provided in the *Presentation Manager Programming Reference*.

Creating Marker Sets and Pattern Sets

Like character sets, marker sets and pattern sets are defined in fonts. For both marker sets and pattern sets, there are system-provided defaults. You also can create your own image markers or patterns, but not outline markers or patterns, using the Font Editor. The font definition containing the image or pattern can be loaded either as a public font (using the **Install** selection from the Control Panel) or as a private font (using `GpiLoadFonts`). The current character mode has no effect on marker sets and pattern sets. Any character from a font can be used as a marker or as an area-fill pattern.

Before you can use a locally defined marker set or pattern set, you must call `GpiCreateLogFont` to define a logical font and to give it an `lcid`, just as you would for a character font. Most of the values you can specify with `GpiCreateLogFont` are not applicable to marker sets and pattern sets. Furthermore, in the instance of marker sets and pattern sets, you usually know that you want to use a particular font.

To select a locally defined marker set or pattern set:

1. Call `GpiQueryFonts` to obtain a match value for the font.
2. Call `GpiCreateLogFont`, specifying the face name, match value, and `lcid` for the font.
3. Select the logical font as the current marker set or pattern set by calling `GpiSetMarkerSet` or `GpiSetPatternSet`, as appropriate.

To revert to using the default pattern or marker sets, call `GpiSetPatternSet` or `GpiSetMarkerSet` with the value `LCID_DEFAULT`.

Using Bit Maps as Area-Fill Patterns

An alternative to using the Font Editor to create your own pattern sets is to use a bit map as an area-fill pattern. You must call `GpiSetBitmapId` to give the bit map an `lcid`, and then call `GpiSetPatternSet`, specifying the `lcid` of the bit map. The bit map becomes a pattern set containing one pattern, and the current pattern-symbol attribute is ignored. `GpiQueryBitmapHandle` returns the handle of the bit map currently known by the `lcid` that you supply as input.

PM cannot distinguish among the `lcids` associated with a particular presentation space. Any `lcid` can identify a character set, marker set, pattern set, or bit map (that is being used as an area-fill pattern). For example, if you call `GpiQueryNumberSetIds` to determine how many `lcids` are currently assigned in the presentation space, the number returned to you includes *all* logical fonts defined for this presentation space. You also can use `GpiDeleteSetId` to remove `lcids` associated with the non-font entities. If you call this function to delete a bit map that is being used as an area-fill pattern, the bit map itself is not deleted; however, it is disassociated from its `lcid`, and the `lcid` can be reused.

Using Paths with Fill Patterns

Using paths lets you fill a character string with a complicated or nonrepeating pattern. The following sequence is recommended:

- Select the required outline font, and set the required character attributes.
- Create a path by calling `GpiBeginPath`, `GpiCharString`, and `GpiEndPath`.
- Set the required pattern symbol and other `AREABUNDLE` pattern attributes.
- Fill the path using `GpiFillPath`.

Figure 9-13 provides an example of using paths with fill patterns to create characters.



Figure 9-13. Using Clip Paths to Create Characters

Summary

Table 9-7 summarizes the font functions.

Table 9-7 (Page 1 of 2). Font Functions	
Function Names	Description
GpiCharString	Draws a character graphics string from the current font from the current position.
GpiCharStringAt	Draws a character graphics string from the current font from a specified position.
GpiCharStringPos	Draws a character string with the first character at the current position and permits positioning of every following character.
GpiCharStringPosAt	Draws a character string and permits positioning for the first and every following character.
GpiCreateLogicalFont	Defines a logical font with specified font attributes.
GpiDeleteSetId	Deletes a logical font.
GpiLoadFonts	Loads a dynamic-link library containing font resources and makes these available as private fonts.
GpiQueryCharBreakExtra	Determines the current character break extra attribute.
GpiQueryCharExtra	Determines the current character extra attribute.
GpiQueryCharSet	Determines the current character set attribute.
GpiQueryCharStringPos	Determines the position of each character for an equivalent <code>GpiCharStringPos</code> without actually drawing any characters.
GpiQueryCharStringPosAt	Determines the position at which each character for an equivalent <code>GpiCharStringPosAt</code> without actually drawing any characters.

Table 9-7 (Page 2 of 2). Font Functions

Function Names	Description
GpiQueryCp	Determines the current code page.
GpiQueryFaceString	Generates a compound face name for a font.
GpiQueryFontAction	Determines if available fonts have been affected since the last time the function was called.
GpiQueryFontFileDescriptions	Determines the family name and face name of each font in the locally defined font file.
GpiQueryFontMetrics	Determines the metric values of the current font.
GpiQueryFonts	Determines the font metrics of every font available with a specified face name.
GpiQueryKerningPairs	Determines kerning-pair information from the current logical font.
GpiQueryLogicalFont	Determines the font attributes (FATTRS) of the specified or current logical font.
GpiQueryNumberSetIds	Determines the number of lcids that are currently assigned in the presentation space.
GpiQuerySetIds	Determines the lcids currently assigned to logical fonts in the specified presentation space.
GpiQueryTextBox	Determines the coordinates of the corners of a text box for an equivalent GpiCharStringPos without actually drawing any characters.
GpiQueryWidthTable	Determines the width of all characters in the current font.
GpiSetCharBreakExtra	Sets the current character break extra attribute.
GpiSetCharExtra	Sets the current character extra attribute.
GpiSetCharSet	Sets the current character set attribute.
GpiSetCp	Assigns a new code page.
GpiUnloadFonts	Unloads a font file (private fonts).

Table 9-8 summarizes the data structures used by the font functions.

Table 9-8. Font Structures

Structure	Description
CHARBUNDLE	Contains the character attributes.
FATTRS	Contains the font attributes of the specified font named in GpiCreateLogFont.
FONTMETRICS	Contains all the font metrics returned by the system to describe a particular font.

Chapter 10. Paths

A *path* is a graphics object composed of one or more figures drawn with graphic primitives. A path can be used for defining a filled area, a complex clipping shape, an outline, and for drawing geometric wide lines.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Line and arc primitives
- Area primitives
- Color and mix attributes
- Regions
- Clipping.

About Paths

Paths provide several useful drawing techniques including:

- An alternate means of generating filled or outlined irregular figures and nonrectangular shapes.
- An efficient means of producing hollow text in an outline font.
- The only means of *clipping* (restricting the area of interest of a picture) to circular, elliptical, or other nonrectangular figures.
- The only means of drawing lines with a thickness that can be transformed.

These techniques are respectively illustrated in Figure 10-1.



Figure 10-1. Drawing Techniques Achieved with Paths

A path is *similar* to an area, but there are fundamental differences between the two. Table 10-1 describes their similarities and contrasts.

Table 10-1 (Page 1 of 2). Comparison of Paths and Areas	
Paths	Areas
Functions that define a path are bracketed by GpiBeginPath and GpiEndPath.	Functions that define an area are bracketed by GpiBeginArea and GpiEndArea.
Paths are defined in world coordinates.	Areas are defined in world coordinates.

Table 10-1 (Page 2 of 2). Comparison of Paths and Areas	
Paths	Areas
Paths can be modified before being displayed or used for clipping.	Areas are displayed immediately on completion.
Paths can be used by applications to perform clipping.	Areas cannot be used as clipping regions.
Paths can be converted into graphics objects called regions.	Areas cannot be converted into regions.
Six operations can be performed on paths, each requiring a separate function: • Outline • Fill • Modify • Stroke • Convert to clip path • Convert to region	Only two operations can be performed on areas (specified with options when creating the area): • Outline • Fill

Path Attributes

Paths do not have a separate xxxBUNDLE data structure associated with them; their attributes are defined by LINEBUNDLE and AREABUNDLE. As described in “Attributes of Line and Arc Primitives” on page 3-2 however, there are certain line attributes that apply only to *geometric* lines, or lines created with some path operations. Depending on the purpose of a path, only certain path attributes are in effect. These and the other path attributes are described in this chapter.

When an application creates a presentation space, the path attributes are set to the default values shown in Table 10-2.

Table 10-2 (Page 1 of 2). Path Attribute Default Values		
Attribute	Default Value	Function that Redefines Attribute
Geometric width	None	GpiSetLineWidthGeom
End	LINEEND_FLAT	GpiSetLineEnd
Join	LINEJOIN_BEVEL	GpiSetLineJoin
Width	1.0	GpiSetLineWidth
Type	LINETYPE_SOLID	GpiSetLineType
Color	CLR_NEUTRAL	GpiSetAttrs (LBB_COLOR)
Mix mode	FM_OVERPAINT	GpiSetAttrs (LBB_MIX_MODE)
Pattern symbol	Solid	GpiSetPattern
Pattern set	LCID_DEFAULT	GpiSetPatternSet
Reference point	(0,0)	GpiSetPatternRefPoint
Foreground color	CLR_NEUTRAL (black on most devices)	GpiSetAttrs (ABB_COLOR)

Table 10-2 (Page 2 of 2). Path Attribute Default Values

Attribute	Default Value	Function that Redefines Attribute
Background color	CLR_BACKGROUND (white on most devices)	GpiSetAttrs (ABB_BACK_COLOR)
Foreground mix	FM_OVERPAINT	GpiSetAttrs (ABB_MIX_MODE)
Background mix	BM_LEAVEALONE	GpiSetAttrs (ABB_BACK_MIX_MODE)
Note: Geometric width, End, and Join take effect only when a path is converted to a geometric wide line using GpiStrokePath or GpiModifyPath. Width, Type, Color, and Mix mode take effect only when a path is outlined using GpiOutlinePath. Pattern symbol, Pattern set, and Reference point take effect only when a path is filled using GpiFillPath or GpiStrokePath.		

Line Width and Geometric Width for Paths

Cosmetic lines represent the mathematical ideal of a 1-dimensional line; the width attribute associated with them is provided only for ease of drawing these lines larger than 1 pel.

In contrast, *geometric* line width is not an attribute; it is a geometric property. To set the width of a geometric line, an application can use GpiSetLineWidthGeom. This geometric width takes effect only when a path is converted to a geometric wide line using GpiStrokePath or GpiModifyPath.

If a geometric line is drawn before the geometric width is specified, the drawn line is defined by the cosmetic line width—usually 1—which results in the thinnest possible line for the currently associated device. A geometric line width has no default in the same way that the sides of a box, drawn with GpiBox, have no length until specified by the function.

The value specified with GpiSetLineWidthGeom is the thickness of the line in world coordinates, and it is subject to scaling by the current transformations in effect at the time GpiModifyPath or GpiStrokePath is issued. For example, if you apply a transform with a scaling factor of 0.5, for an object whose current geometric line width is 4 world coordinates, the width of the displayed line will be halved.

In the floor plan shown in Figure 10-2, the outer walls were drawn using geometric lines. All of the other objects were drawn using normal lines.

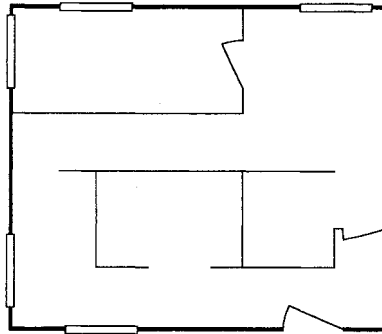


Figure 10-2. Geometric Lines and Normal Lines

Your application can determine the current geometric width by issuing `GpiQueryLineWidthGeom`.

Line End

The *line end* attribute specifies the shape of the unattached end of a geometric line. Lines whose shapes are partially defined by a geometric width have to be closed, unlike cosmetic lines that simply end. An application can draw geometric lines with square, flat, or round ends.

Note: The end attribute takes effect only when a path is converted to a geometric width line using `GpiStrokePath` or `GpiModifyPath`.

Set the current geometric line end attribute with `GpiSetLineEnd`. The attribute applies to all subsequent unattached lines within the path bracket. Table 10-3 describes the three standard line ends provided by the PM programming interface; you cannot define your own end types.

Table 10-3. Standard Geometric Line End Types		
Type	Identifier	Long Value
Flat, flush with line end	LINEEND_FLAT	1L
Round, past line end	LINEEND_ROUND	2L
Flat, but extends past line end	LINEEND_SQUARE	3L

The default line end type (`LINEEND_DEFAULT`) is identical to the `LINEEND_FLAT` type, and has a long value of 0L. The error linetype (`LINEEND_ERROR`) has a long value of -1L.

Figure 10-3 illustrates these three types of line ends. Your application can determine the current geometric line end by issuing `GpiQueryLineEnd`.

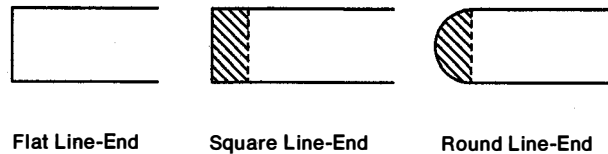


Figure 10-3. Closing Unattached Geometric Lines

A square line end extends the line by a distance that is half the width of the line. For example, if the line is 6 coordinate units wide, a square line end extends it by 3 coordinate units.

A round line end is constructed by drawing a circle whose radius is half the width of the line.

Line Join

The *line join* attribute specifies the shape formed by two intersecting geometric lines. Where one wide line joins another, the nature of the join must be defined. An application can select a beveled, rounded, or mitred line-join style.

Note: The join attribute takes effect only when a path is converted to a geometric wide line using `GpiStrokePath` or `GpiModifyPath`.

Set the current geometric line-join attribute with `GpiSetLineJoin`. The attribute applies to all subsequent intersection of lines within the path bracket. Table 10-4 describes the three standard line joins provided by the PM programming interface. You cannot define your own join types.

Table 10-4. Standard Geometric Line Join Types		
Type	Identifier	Long Value
Diagonal corner	LINEJOIN_BEVEL	1L
Rounded corner	LINEJOIN_ROUND	2L
90° angled corner	LINEJOIN_MITRE	3L

The default join type (`LINEJOIN_DEFAULT`) is identical to the `LINEJOIN_BEVEL` type, and has a long value of 0L. The error linetype (`LINEJOIN_ERROR`) has a long value of -1L. Figure 10-4 illustrates these three types of line joins. Your application can determine the current geometric line join using `GpiQueryLineJoin`.

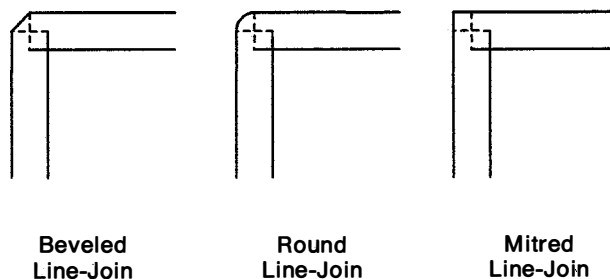


Figure 10-4. Joining Wide Lines

Cosmetic Line Attributes for Paths

When a path is drawn as a cosmetic line with `GpiOutlinePath`, the following attributes are defined by the `LINEBUNDLE` data structure:

- Line width
- Line type
- Line color
- Line mix.

These attributes are described in “Attributes of Line and Arc Primitives” on page 3-2.

These attributes follow the current definitions and appear just as line primitives do. Since the line primitive has only a foreground color and mix attribute, the current color of the drawing surface affects the appearance of these paths more than it does the appearance of paths that define geometric lines. Those paths follow the area attributes and are affected by background color and background mix attributes as well.

Area Attributes for Paths

When a path is drawn as a geometric line with `GpiStrokePath` or `GpiModifyPath`, the following attributes are defined by the `AREABUNDLE` data structure:

- Pattern symbol
- Pattern set
- Pattern reference point
- Area foreground color
- Area background color
- Area foreground mix
- Area background mix.

These attributes are discussed in “Attributes of Area Primitives” on page 5-1.

Area attributes follow the current definitions and appear just as area primitives do. The operating system uses the pattern symbol to fill the interior of the path that defines the geometric line. Any alterations to the fill pattern or reference point affect the appearance of a geometric line just as it does an area.

Path Color and Mix Attributes

The color attribute defines the color used to draw a primitive or an object. The mix attribute determines how the color of a primitive or an object is combined with the color of the drawing surface, or any other objects on the surface. Both attributes also are described in Chapter 7, “Color and Mix Attributes.” Unlike the color and mix attributes for primitives, and certain other objects, the path color is defined by different attributes in different situations as illustrated in Figure 10-5.

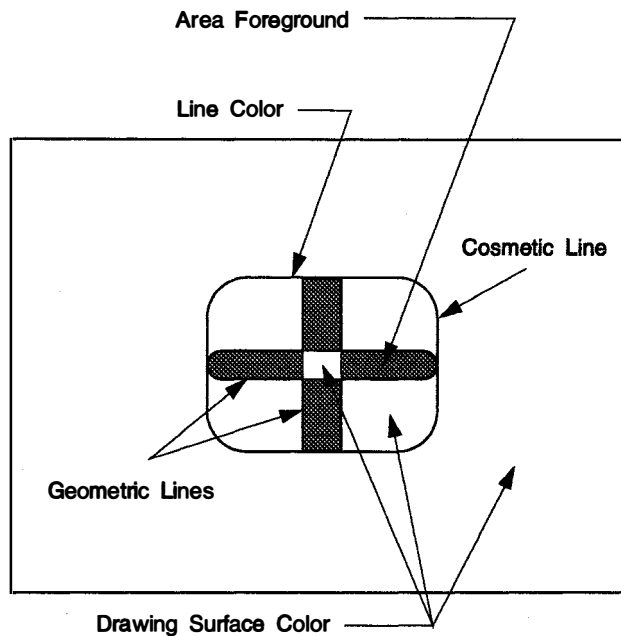


Figure 10-5. Path Objects. Path colors are determined by the area color, area background color, area mix, and area background mix attributes if the path is stroked into a wide line. The cosmetic line path color is determined by the line color and mix attributes.

The line color defines the color used to draw the output from any of the OS/2 operating system line functions issued within the path. The line colors can change within the path bracket, just as they can within area brackets. The interaction of the line with the drawing-surface color is controlled by the line mix attribute, which also can vary within the path bracket.

After the path is closed, the path can be *stroked* into a geometric (or wide) line. Depending on the construction options, described in “Path Fill” on page 10-11, the geometric line can be filled. The appearance of the pattern symbol used to fill the path depends on the area attributes. The definitions of the area color and background color depend on the pattern symbol as described in “Area Colors and Mix Attributes” on page 5-5.

To specify a new color or mix attribute refer to the instructions in “Line Color and Mix Attributes” on page 3-5 for lines, or in “Area Colors and Mix Attributes” on page 5-5 for areas.

Path Brackets

The functions that create and define the path always are bracketed by the `GpiBeginPath` and `GpiEndPath` functions. Only one path can be defined between these two functions. However, within the path, there can be any number of figures.

`GpiBeginPath` signals the start of a path definition. Each path you define becomes the *current path* and replaces any existing path in the presentation space. The *current position* is not changed by `GpiBeginPath`, although it is updated by any drawing instructions that follow.

`GpiBeginPath` accepts as input only a *path identifier* of 1; this identifier is used by the functions performing path operations to identify the path.

Due to the number of different operations that can be performed on paths, the path definition can contain both open and closed figures. If a figure does not start and end at the same coordinate point, it is classified as an *open* figure. Unlike an area bracket, if you issue `GpiSetCurrentPosition` or `GpiMove` within a path definition, the current figure is *not* automatically closed. Other functions that merely separate figures within a path definition are listed in the *Presentation Manager Programming Reference*.

The PM programming interface provides a method of closing figures within a path. The function `GpiCloseFigure` (valid only in path brackets) closes a figure in the path, whose start and end points are not the same, by drawing a straight line from the current position to the start position of the figure. Use `GpiCloseFigure` if you want the line join attribute to be applied between the start and end point of a figure. Do not use it if you want the line end attributes to be employed as the start and end points for a geometric wide line.

If the path definition contains full arc or box primitives, however, you must not close them using `GpiCloseFigure`. `GpiBox` and `GpiFullArc` generate completely closed figures and must not be used within another figure definition.

To signal the end of the current path definition, issue `GpiEndPath`. The path definition is constructed in the currently associated presentation space. The current position always is updated to the end point of the last line drawn.

The functions that generate a path are enclosed in a path bracket. Applications can use the functions in the following list within a path bracket.

- GpiBeginElement
- GpiBox
- GpiCallSegmentMatrix
- GpiCharString
- GpiCharStringAt
- GpiCharStringPos
- GpiCharStringPosAt
- GpiCloseFigure
- GpiComment
- GpiCreateLogFont
- GpiDeleteSetId
- GpiElement
- GpiEndElement
- GpiEndPath
- GpiFullArc
- GpiGetData
- GpiLabel
- GpiLine
- GpiMarker
- GpiMove
- GpiOffsetElementPointer
- GpiPartialArc
- GpiPointArc
- GpiPolyFillet
- GpiPolyFilletSharp
- GpiPolyLine
- GpiPolyMarker
- GpiPolySpline
- GpiPop
- GpiPutData
- GpiQueryArcParams
- GpiQueryAttrMode
- GpiQueryCurrentPosition
- GpiSetArcParams
- GpiSetAttrMode
- GpiSetAttrs
- GpiSetCharAngle
- GpiSetCharBox
- GpiSetCharDirection
- GpiSetCharMode
- GpiSetCharSet
- GpiSetCharShear
- GpiSetColor
- GpiSetCp
- GpiSetCurrentPosition
- GpiSetEditMode
- GpiSetLineEnd
- GpiSetLineJoin
- GpiSetLineType
- GpiSetLineWidth
- GpiSetMarker
- GpiSetMarkerBox
- GpiSetMarkerSet
- GpiSetMix
- GpiSetModelTransformMatrix

Warning: Some GPI functions cannot be issued while an open path definition exists. In particular, you cannot include image or area primitives in a path definition.

Path Operations

The operations performed on a path determine which of the path attributes become applicable. Unlike specifying area brackets, the variety of operations also calls for separate functions instead of specifying options with GpiBeginPath. Specifically, the following six operations can be performed on a path:

- Outline
- Fill
- Modify
- Stroke
- Convert to clip path
- Convert to region.

There are other graphics operations provided by the PM programming interface, for example, transformations; however, they can apply to entities other than paths.

Upon completion, the GpiFillPath, GpiOutlinePath, GpiStrokePath, and GpiPathToRegion functions delete the current path definition. This path is not available for any subsequent operations.

GpiModifyPath does not delete the current path definition. An application can modify a path before issuing one of the aforementioned functions.

The first time your application issues `GpiSetClipPath`, the current path definition is converted into the current clip path. `GpiSetClipPath` deletes the path definition upon completion but not the clip path definition.

A path definition can be stored in a graphics segment; and, if that segment is retained, the path can be re-created as required. Segments are discussed in Chapter 12, "Creating and Drawing Retained Graphics."

Path Outline

To draw the outline of the current path, issue `GpiOutlinePath`. This function accepts a path identifier (which must be 1) as input. `GpiOutlinePath` does not create a geometric line; therefore, the cosmetic line attributes are used.

`GpiOutlinePath` normally has the same effect as though the lines and arcs used to create the path were drawn without actually being part of the path. Any open figures within the path are not closed automatically. If the path contains character strings that use an outline font, the characters are not filled.

The following functional sequence is recommended for drawing outline text or outline figures:

<i>Table 10-5. Functional Sequence for Drawing Outline Text and Figures</i>	
Function Name	Effect
GpiBeginPath	Starts the path definition.
GpiMove	Moves to the starting point of the first figure.
GpiCharString, GpiFullArc	Uses an outline font to specify text or define the lines in a figure.
GpiEndPath	Ends the path.
GpiOutlinePath	Draws the text or figure outlines.

If you are drawing a complex path, consisting of several character strings that use an outline font, it can take quite some time to produce. If you want to create a only rough draft, you might find it quicker to draw an outline instead of the filled path. This improves performance and might be acceptable for small characters. When they are small enough, hollow characters can look like filled characters.

You can create hollow characters in either of two ways:

- Outside a path, issue `GpiCreateLogFont`.
This enables you to define an outline font that uses hollow characters.
- Inside a path, draw the characters, then display the path with `GpiOutlinePath`.
Only the character outlines contribute to the path definition even if the font is not hollow. (This might cause ambiguous results.)

Using outlined paths to create character outlines also can be used to define a clip path in the shape of a letter or word.

Path Fill

To draw the current path and fill its interior using the current pattern symbol, issue `GpiFillPath`. The path is deleted after the interior is filled. The boundary lines are considered part of the path interior and are drawn with this function. Any open figures in the path definition are closed automatically by `GpiFillPath`.

This function accepts the path identifier (which must be 1) as input, and either of two construction options:

- `FPATH_ALTERNATE` (the default)
- `FPATH_WINDING` (must be selected if the path has been modified).

Paths, like area primitives, can be filled in alternate mode or winding mode. If the path consists of multiple, intersecting figures, the path-fill mode affects the final appearance of the path.

Figure 10-6 shows two identical paths that were filled, each using one of the two modes. Each path consists of a triangle drawn within a rectangle. The path on the left was filled using the alternate mode. The path on the right was filled using the winding mode.

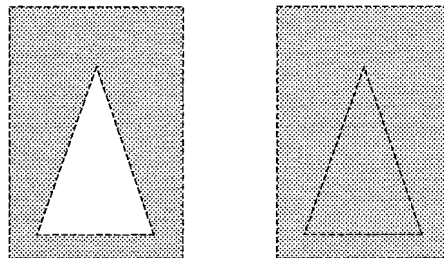


Figure 10-6. Alternate and Winding Fill Modes

Paths in Alternate Mode

For paths drawn in alternate mode (as with area primitives), the following is true:

- If you have to cross an odd number of boundaries in the path when drawing an imaginary line parallel to the x-axis, a point is included in the filled path from that point to positive infinity.
- If you have to cross an even number of boundaries in the path when drawing an imaginary line parallel to the x-axis, a point is not included in the filled path from that point to positive infinity.

Figure 10-7 on page 10-12 shows how the operating system determines the filled portion for the path shown in Figure 10-6. The path outside the triangle, but inside the rectangle, is filled, because the imaginary lines drawn from those points in the positive x-direction intersect the path boundaries an odd number of times.

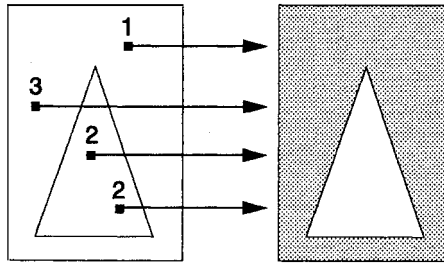


Figure 10-7. Calculating Filled Paths Constructed in Alternate Mode. Parts of the path with odd tallys are filled; parts with even tallys are not filled.

Paths in Winding Mode

For paths drawn in winding mode (also like area primitives), the direction the boundaries in the path are drawn in determines whether a given point is included in the filled path. The drawn direction of a boundary line depends on both the graphics functions used to draw it and the world coordinates that define it.

For box primitives, for example, if the specified diagonal corner is to the right, to the top, or both, of the current position, the box is drawn counterclockwise. If the specified diagonal corner is to the left, to the bottom, but not both, the box is drawn clockwise.

For a polyline primitive, the direction of the boundary depends on the relative values of the start and end points of each line. For a polygon primitive, the direction of the boundary depends on the relative values of the specified vertices.

To determine whether a given point is included in the filled path, count the number of boundary lines to be crossed to move from that point to infinity. For each boundary line drawn in one direction add one to the tally. For each boundary line drawn in the opposite direction, subtract one from the tally. A point is within the area if the result is nonzero. As with the alternate mode, the imaginary determining line is drawn in the positive x-direction from the point in question toward infinity.

Figure 10-8 shows how the operating system determines the filled portion for the path shown in Figure 10-6 on page 10-11. Assume that both the rectangle and triangle were drawn in the same direction, whether clockwise or counterclockwise is immaterial. Both figures are filled, because the number of times the imaginary lines drawn from those points in the positive x-direction intersect the path boundaries is continuously summed. There is never a subtraction of a boundary tally to reduce the total to 0.

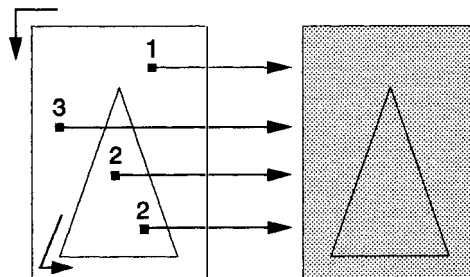


Figure 10-8. Paths Constructed in the Same Direction in Winding Mode. Parts of the path with nonzero tallys are filled; parts with zero tallys are not filled.

If two figures, for example in Figure 10-9, are drawn in different directions, the tally for the inner triangle is 0 and the area looks exactly as it does in alternate mode.

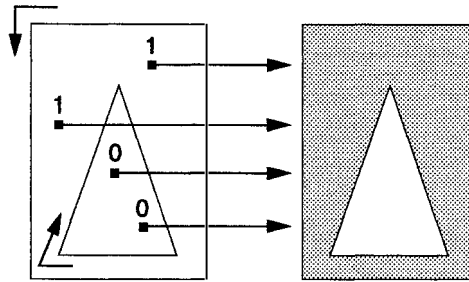


Figure 10-9. Paths Constructed in Different Directions in Winding Mode. Parts of the path with nonzero tallies are filled; parts with zero tallies are not filled.

Path Modification

To convert the current path to one that describes the envelope of a geometric wide line, issue `GpiModifyPath`. The current geometric line attributes are then applied to the figures within the path. The line end, line join, and geometric line width attributes all must be set before modify-path or stroke-path operations begin, because it is during those times that the attributes are applied. Cosmetic line attributes of width and type have no effect on geometric lines.

`GpiModifyPath` accepts the path identifier (which must be 1) and the modification option `MPATH_STROKE` as input.

Subsequently, the modified path can be filled to display the geometric wide line, but only in winding mode. Alternatively, the modified path can be converted into a clip path, but again, only in winding mode.

Open figures within the path are not closed automatically. Figures not explicitly closed with `GpiCloseFigure` are drawn with line ends rather than line joins at their start and end points. If the figures within the path overlap, the geometric width envelope compensates so that the overlap portion is not drawn blank in XOR or exclusive-OR mode.

Now the geometric lines can be scaled.

Note: The current transforms are applied to the primitives inside a path bracket when the path is defined. This binds the path definition in device coordinates at that time. The path is unaffected by subsequent transformations, except for those (such as scaling) that affect the *width* of geometric (wide) lines. Since the geometric line width attribute is not applied until the path is converted into a wide line by `GpiModifyPath` or `GpiStrokePath`, the width of geometric lines is unaffected by earlier transformations directed at the path definition.

After creating a path bracket, geometric wide lines can be constructed by either:

- Stroking the path (`GpiStrokePath`)
- Modifying the path (`GpiModifyPath`), then filling the path (`GpiFillPath`).

`GpiStrokePath` is slightly more efficient, but `GpiModifyPath` and `GpiFillPath` offer more flexibility (by way of the fill options). After drawing the lines no alterations can be made.

Path-Stroking

To modify and fill the current path in a single operation, issue `GpiStrokePath`. Then the path is drawn immediately to the output device, with the current geometric line attributes applied to the figures within the path. Certain device drivers can use this function to optimize storage.

`GpiStrokePath` automatically fills the path in winding mode with the current area pattern symbol. When `GpiStrokePath` is complete, the path definition is deleted from the device context.

`GpiStrokePath` is subject to the same constraints as functions that perform simple line modification:

- The line end, line join, and geometric line width attributes all must be set before stroke operations.
- Cosmetic line attributes of width and type have no effect.
- Any open figures within the path are not closed automatically.
- If figures are not explicitly closed with `GpiCloseFigure`, they are drawn with line ends rather than line joins at their start and end points.
- If the figures within the path overlap, the geometric width envelope compensates so that the overlap portion is not drawn blank when drawn in XOR (exclusive-OR) mode.

This function accepts as input both the path identifier (which must be 1) and the stroke option (which must be 0L).

When the operating system strokes a path, it draws a geometric line of specified width along the original figure that defined the path; then, fills the wide line. If the original figure is not a closed shape, the operating system does not automatically close it before filling the path. Figure 10-10 shows the effects of `GpiStrokePath` on a box originally defined with normal (cosmetic) lines.

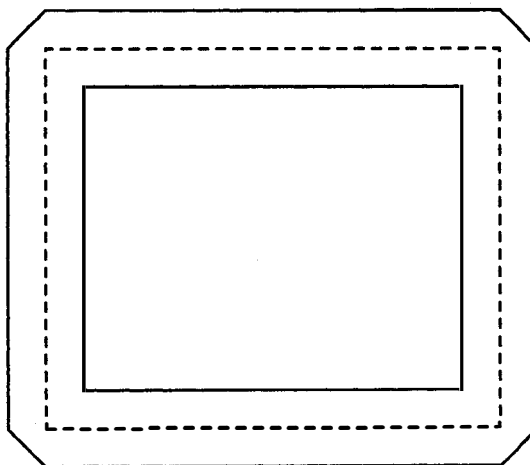


Figure 10-10. Defining Lines with a Geometric Width. The broken line is the figure defined within the path. The solid lines show the path after it has been converted. Each line has a geometric width of n coordinate units, and the line joins have been defined as beveled.

After setting these attributes, the application can draw the line with `GpiStrokePath`. After the lines are drawn, an application can scale stroked paths with a scaling transformation.

As an alternative to the `GpiStrokePath`, you can convert the path using `GpiModifyPath`, which does *not* draw the path on the current output device. To draw a modified path, issue `GpiFillPath`, which draws the path and fills it with the current area-fill pattern in winding mode. A modified path cannot be filled in alternate mode.

On some devices, it might be that the `GpiStrokePath` method works better than `GpiModifyPath`, followed by `GpiFillPath`.

Path Conversion to Clip Path

Clipping is the process an application uses to limit graphics output to a specific area (called the *clipping area*) of the display or page. Clipping is described further in Chapter 16, "Clipping and Boundary Determination."

There are several clipping functions provided by the PM programming interface; however, if your application requires an irregular complex shape for a clipping area, it must define the shape with a path. To convert the path into a clipping boundary, issue `GpiSetClipPath`. The clip path, as defined by this operation, becomes the current clip path for all subsequent drawing.

This function accepts a path identifier and one of two construction options as input:

- `SCP_ALTERNATE` (default)
- `SCP_WINDING` (must be selected if the path has been modified).

Unlike the path operations described previously, `GpiSetClipPath` accepts two different path identifiers:

- 1
- 0 (default).

The default path identifier of 0, called `SCP_RESET`, resets the clip path to infinity, which displays the picture without clipping. If this value is selected, the current clip path definition is discarded instead of stored.

A path identifier of 1 is called `SCP_AND`. For `GpiSetClipPath`, a path identifier of 1 causes the clip path to be redefined as the mathematical intersection of the stored clip path and the current path definition. For all other path operations, an identifier of 1 specifies the current path as the recipient of the operation. The only method of specifying the clip path as the current path, after `GpiSetClipPath` has been issued, is to issue `GpiSetClipPath` again: the first issuance with a path identifier of 0; the second, with a path identifier of 1. The path identifiers and the construction mode can be ORed together for certain effects.

Any open figures within a path are closed automatically. The boundaries of the path are considered part of the interior, so any point on the boundary is not clipped. Figure 10-11 shows the result of clipping text with a triangular clip path.

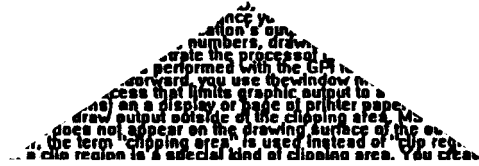


Figure 10-11. Triangular Clip Path

Path Conversion to Region

To convert the current path to a region, issue `GpiPathToRegion`. Then, the new region can be modified using any of the region functions that allow the creation of irregular shaped regions. For detailed information about regions, see Chapter 11, "Regions."

Any open figures in the path definition are closed automatically by `GpiPathToRegion`. Upon conversion, the current path definition is discarded just like it is when being converted to a clip path. When `GpiPathToRegion` is complete, the path definition is deleted automatically.

This function accepts the path identifier (which must be 1) and one of two construction options as input:

- `FPATH_ALTERNATE` (default)
- `FPATH_WINDING` (must be selected if the path has been modified).

Using Paths

You can use path functions to:

- Draw geometric lines and filled polygons
- Draw outline text
- Create a triangular clip path.

Drawing a Geometric (Wide) Line

Figure 10-12 is an example of how to draw a straight line, 10 units wide with rounded ends.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
void fncPATH01(void){
    POINTL pt1;
    HPS hps;

    GpiSetLineWidthGeom(hps, 10L);          /* Sets line width to ten          */
    GpiSetLineEnd(hps, LINEEND_ROUND);      /* Sets line end to round          */
    GpiBeginPath(hps, 1L);                  /* Begins path                      */
    pt1.x = 7;
    pt1.y = 15;
    GpiMove(hps, &pt1);                    /* Sets current position            */
    pt1.x = 450;
    pt1.y = 15;
    GpiLine(hps, &pt1);                    /* Draws line                       */
    GpiEndPath(hps);                        /* Ends path                        */
    GpiStrokePath(hps, 1L, 0L);             /* Draws geometric line             */
} /* fncPATH01 */
```

Figure 10-12. Drawing a Wide Line

Drawing Filled Polygons

Figure 10-13 is an example of how to draw an empty triangle within a solid rectangle.

```
#define INCL_GPIPATHS
#include <os2.h>
void fncPATH02(void){
    HPS hps;
    POINTL apt11[4] = {                    /* Array of points for triangle */
        50, 50,
        100, 100,
        150, 50,
        50, 50 };

    POINTL apt12[5] = {                    /* Array of points for rectangle */
        25, 25,
        25, 200,
        200, 200,
        200, 25,
        25, 25 };

    GpiBeginPath(hps, 1L);                 /* Begins path                    */
    GpiMove(hps, apt11);                   /* Sets current position          */
    GpiPolyLine(hps, 4L, apt11);           /* Plots points for triangle      */
    GpiMove(hps, apt12);                   /* Sets current position          */
    GpiPolyLine(hps, 5L, apt12);           /* Plots points for rectangle     */
    GpiEndPath(hps);                       /* Ends path                      */
    GpiFillPath(hps, 1L, FPATH_ALTERNATE); /* Draws triangle and rectangle  */
} /* fncPATH02 */
```

Figure 10-13. Drawing Filled Polygons

Drawing Outline Text

Figure 10-14 shows an example of how to draw text from an outline font with `GpiOutlinePath`.

```
#define INCL_GPIPRIMITIVES
#include <os2.h>
void fncPATH03(void){
    LONG lcid = 1;                /* Identifier for font */
    FATTRS fattrs;                /* Structure for font attributes */
    SIZEF sizfx;                  /* Structure for character box */
    POINTL ptl;                   /* Structure for starting point */
    HPS hpsWin;

    fattrs.usRecordLength = sizeof(FATTRS);
    fattrs.fsSelection = FATTR_SEL_OUTLINE; /* Specifies outline font */
    fattrs.lMatch = 0;             /* System determines font */
    fattrs.szFacename[0] = 0;
    fattrs.idRegistry = 0;
    fattrs.usCodePage = 0;
    fattrs.lMaxBaselineExt = 0;
    fattrs.lAveCharWidth = 0;
    fattrs.fsType = 0;
    fattrs.fsFontUse = FATTR_FONTUSE_OUTLINE; /* Specifies outline font */

    GpiCreateLogFont(hpsWin, (PSTR8) NULL, lcid, &fattrs);
    GpiSetCharSet(hpsWin, lcid); /* Sets logical font */

    sizfx.cx = MAKEFIXED(30, 0); /* Width of character box */
    sizfx.cy = MAKEFIXED(30, 0); /* Height of character box */
    GpiSetCharBox(hpsWin, &sizfx) /* Sets size of character box */

    GpiBeginPath(hpsWin, 1L); /* Begins path */
    ptl.x = 100;
    ptl.y = 200;
    GpiMove(hpsWin, &ptl); /* Sets current position */
    GpiCharString(hpsWin, 7L, "Outline"); /* Establishes char. string */
    GpiEndPath(hpsWin); /* Ends path */
    GpiOutlinePath(hpsWin, 1L, 0L); /* Draws character string */
} /* fncPATH03 */
```

Figure 10-14. Drawing Outline Text

Creating a Triangular Clip Path

Figure 10-15 shows an example of how to create a triangular clip path; then, write text into it.

```
#define INCL_GPIPATHS
#include <os2.h>
void fncPATH04(void){
    POINTL ptlStart;
    HPS hps;
    LONG i;

    POINTL aptl[4] = {           /* Array of points for triangle */
        35, 45,
        100, 100,
        200, 45,
        35, 45 };

    GpiBeginPath(hps, 1L);       /* Begins path bracket */
    GpiMove(hps, aptl);          /* Sets current position */
    GpiPolyLine(hps, 4L, aptl);  /* Plots points for triangle */
    GpiEndPath(hps)              /* Ends path bracket */

    GpiSetClipPath(hps, 1L, SCP_ALTERNATE | SCP_AND);

    /* Write a block of text. */

    ptlStart.x = 50;
    for (i = 50L; i < 110L; i += 10L) {
        ptlStart.y = i;
        GpiCharStringAt(hps, &ptlStart, 18L, "String of text!");
    } /* for */
} /* fncPATH04 */
```

Figure 10-15. Creating a Triangular Clip Path

Summary

Table 10-6 summarizes the path object functions.

<i>Table 10-6. Path Functions</i>	
Function Name	Description
GpiBeginPath	Begins a path definition.
GpiCloseFigure	Draws a line within the path definition from the current position to the starting position of the figure.
GpiEndPath	Terminates a path.
GpiFillPath	Draws the path, fills it with the current AREABUNDLE attributes, then deletes the path. The fill can be in either winding or alternate mode.
GpiModifyPath	Modifies a path to describe the envelope of a geometric wide line by applying line end, line join, and geometric line width attributes.
GpiOutlinePath	Draws only the borders of a path.
GpiPathToRegion	Converts a path to a region.
GpiSetClipPath	Converts the created path into a clip path. The original path is deleted.
GpiSetLineEnd	Defines the line end attribute.
GpiSetLineJoin	Defines the line join attribute.
GpiSetLineWidthGeom	Defines the geometric line width attribute.
GpiStrokePath	Converts a path to the envelope of a geometric wide line using the line end, line join and geometric line width attributes fills the path using area bundle attributes and then deletes the path.

Table 10-7 summarizes the data structures used by the path object functions.

<i>Table 10-7. Path Structures</i>	
Structure Name	Description
AREABUNDLE	Data structure of area primitive attributes.
LINEBUNDLE	Data structure of line primitive attributes.

Chapter 11. Regions

A *region* is a graphics object usually composed of one or more rectangles. Converted to clip regions, they are used mainly to define a clipping boundary, in device coordinates, for multiple intersecting rectangles. Clip regions provide the required clipping in an update region during WM_PAINT processing when it is necessary to repaint part of a window. Regions also can be used for area fill.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Area primitives
- Color and mix modes
- Coordinate spaces
- Transformations
- Clipping.

About Regions

A region consists of one or more overlapping or separate rectangles in an application's device space. The sides of the rectangles are parallel to the x- and y-axes in the device coordinate space. An irregular, nonrectangular path can be converted into a region, as explained in Chapter 10, "Paths." However, unless otherwise specified, the assumption of this chapter is that a region is rectangular.

If a region consists of intersecting rectangles, the intersecting sides are always perpendicular. If the rectangles have no common elements, the region is defined as separate rectangles.

Unlike areas and paths, which are defined in world coordinates, regions are device-dependent and, so, are defined in device coordinates. The device coordinates are *inclusive* (inside the rectangle) at the bottom and left coordinate boundaries, and *exclusive* (outside the rectangle and clipped) at the top and right boundaries.

Each region is created for the device that currently is associated with a presentation space. Also, after creation, regions are available for various operations; a *region handle* identifies the region for subsequent operations.

Figure 11-1 shows a region that consists of two intersecting rectangles. An application defined the region by passing an array, containing the coordinates for the two rectangles, to the `GpiCreateRegion` function. The application then drew the region with `GpiPaintRegion`.

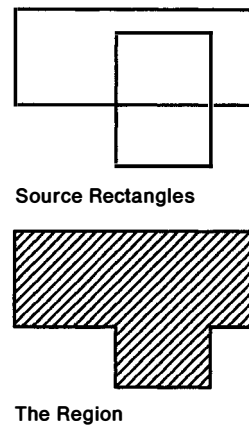


Figure 11-1. Defining a Region. This region comprises two overlapping rectangles.

System Implementation

Regions are device-dependent objects and, therefore, are associated with a particular device. For this reason, when an application specifies a presentation space as input to one of the region functions, that presentation space must be associated with a device context. The associated device context serves to identify the device. Because a region is specific to the device for which it was created, a region should not be created for one device and then used for another.

If the device is a printer, a metafile can be used to store region functions as *escapes*, and the resulting metafile can be displayed on any device. The appearance of the result of any region function is affected by differences in the `pel:aspect ratio` and device resolution on different printers.

Most of the region-oriented functions use the `RECTL` structure to define the device-space coordinates of the rectangle. When an application creates a rectangle in a device space and passes its coordinates to a region function, the operating system excludes the top and rightmost edges of the rectangle. This means that an application must add 1 to the values in the `xRight` and `yTop` fields of the `RECTL` structure to obtain the desired dimensions. For example, if an application requires a region that measures 100-by-100 device units, with a lower-left corner at (10,10), `xRight` and `yTop` should be set to 111 instead of 110.

Region Attributes

Regions do not have a distinct `xxxBUNDLE` data structure associated with them. Unlike primitives or even paths, the purpose of regions is to provide a definition for an operation—not visual output. However, there are two functions that present the region as a visible entity.

GpiPaintRegion paints a region in the presentation space. This function accepts only the region handle as input. GpiPaintRegion does not cause graphics orders to be added to the current segment. Therefore, in retain or draw-and-retain modes, you are advised to use paths or areas rather than regions to ensure the desired effect.

GpiFrameRegion draws a frame around a region by tracing the inner perimeter with a rectangle of a specified size. This function accepts the region handle and the desired frame thickness as input.

A region's visible output is controlled by the following AREABUNDLE attributes:

- Pattern symbol
- Pattern set
- Pattern reference point
- Area foreground color
- Area background color
- Area foreground mix.

These attributes follow the current definitions and appear just as area primitives do. The operating system uses the pattern symbol to fill the interior of the region. Any alterations to the fill pattern or reference point affects the appearance of a region, just as it does in an area.

Because neither the painted region nor the region frame have boundary lines, no LINEBUNDLE attributes influence the appearance of a region resulting from the GpiPaintRegion or GpiFrameRegion functions.

Region Creation

Areas and paths are created and defined within a bracket; regions are defined by a single function, GpiCreateRegion. As input to GpiCreateRegion, supply the total number of, and the coordinates for, each rectangle that contributes to the region. The following applies to region coordinates:

- They define the bottom left and top right coordinates of each rectangle that makes up the region.
- They must be in the range -32768 through +32767.
- They always are device coordinates, because regions always are created and drawn in an application's device space.

The output from GpiCreateRegion is the *region handle*, which identifies the region for subsequent operations. Because each region can be distinctly identified, the region functions allow applications to work with more than one region at a time.

For example, an application can combine two regions with GpiCombineRegion or compare two regions with GpiEqualRegion. GpiQueryRegionRects retrieves the coordinates of the rectangles that make up a region, enabling an application to create a copy of that region. To create the copy, you supply, as input to a new GpiCreateRegion function, the exact rectangles returned from GpiQueryRegionRects.

Region Operations

The following operations can be performed on a region:

- Creating Regions
- Drawing Regions
- Moving Regions
- Determining Region Characteristics
- Converting a Path to a Region
- Converting a Region to a Clip Region
- Deleting Regions.

Creating Regions

To create a rectangular region, call `GpiCreateRegion`. This function accepts, as input, the number of rectangles to be ORed into a single region and the coordinates of those rectangles. If the number of rectangles is 0, an empty region is created.

To create a new region from regions that already exist, call `GpiCombineRegion`. This function accepts, as input, a region handle for the target region, a region handle for each of the 2 source regions, and an options flag that specifies the way the 2 source regions are combined.

Note: The destination region can be either of the 2 source regions; in which case, that region is replaced by the new region.

The rectangles are differentiated by one region's being distinctly "the first source rectangle" and the other region labeled "the second source rectangle." The order of the source rectangles affects the way `GpiCombineRegion` combines them. You can use `GpiCombineRegion` an indefinite number of times to form complex polygons; however, the function is limited to combining only 2 regions each time it is called. The 2 source regions must be created for the same type of device.

Following are the 5 ways in which 2 regions can be combined:

Option	Result
CRGN_OR	Union of the 2 source regions.
CRGN_XOR	Symmetric difference of the 2 source regions.
CRGN_COPY	Source region 1 only; source region 2 is ignored.
CRGN_AND	Intersection of the 2 source regions.
CRGN_DIFF	Source region 1 minus source region 2.

The effects of these different combining operations on overlapping regions are shown in Figure 11-2. Their effects on separate regions are shown in Figure 11-3 on page 11-6.

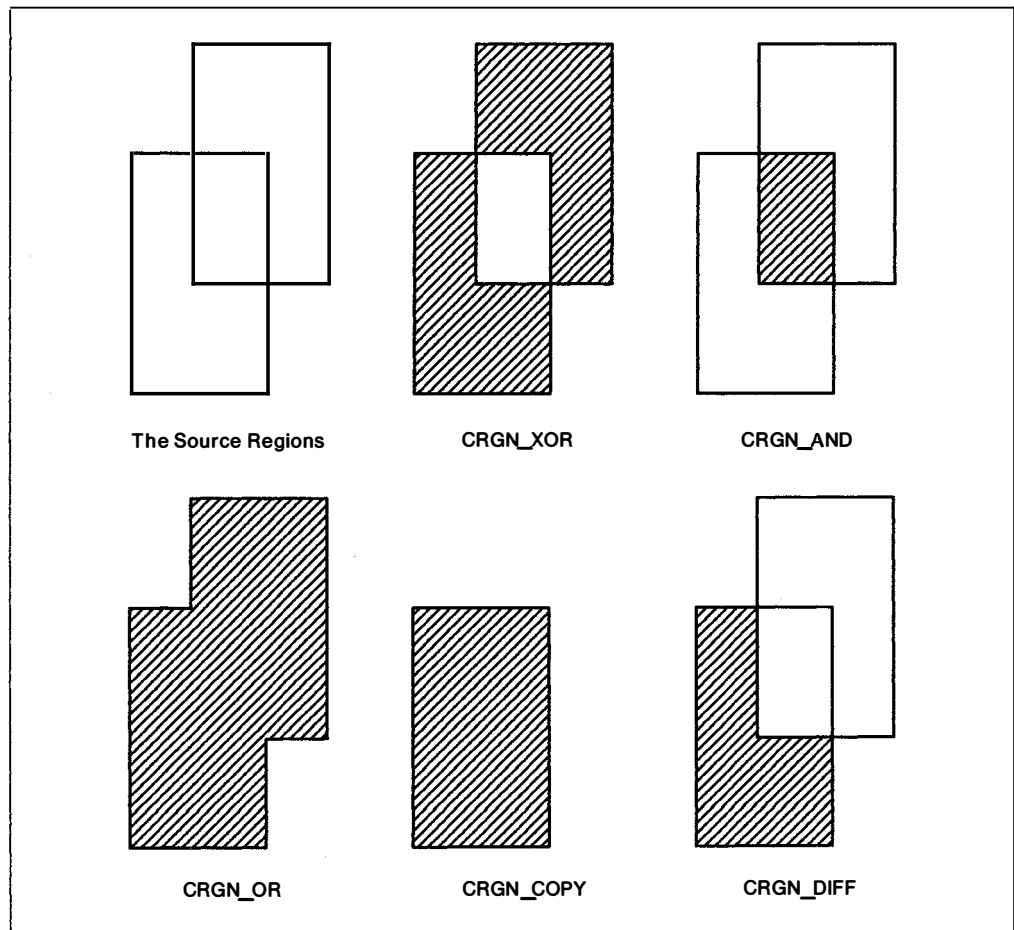


Figure 11-2. Combining Overlapping Regions. The source regions are two overlapping rectangles. The regions that result from the various combine operations are shown as shaded areas.

Output from `GpiCombineRegion` tells you whether the resulting region is a NULL region, rectangular region, or complex region. A *complex region* is any region defined by two or more rectangles.

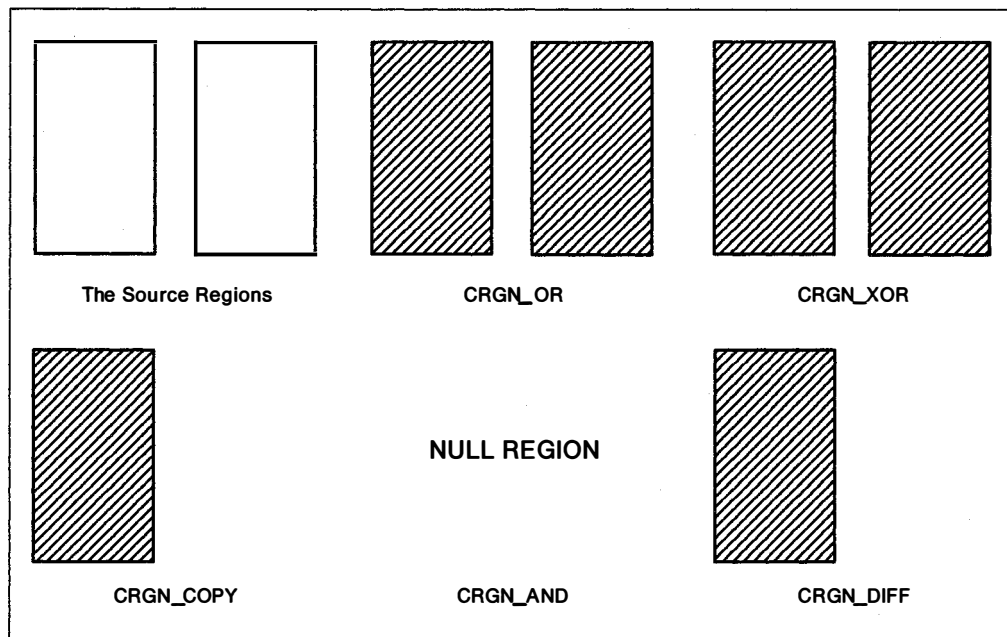


Figure 11-3. Combining Disjoint Regions. The two source regions do not overlap. The regions that result from the various combine operations are shown as shaded areas. The result from a CRGN_AND combining of separate regions is a NULL region.

To change the definition of a region while keeping the same region handle, call `GpiSetRegion`. This function accepts, as input, the number and dimensions of the rectangles that now define the region, just as `GpiCreateRegion` does. However, `GpiSetRegion` also accepts the handle of the region to be updated. `GpiSetRegion`, like `GpiCreateRegion`, takes a region definition to be the union of a sequence of rectangles (that are effectively ORed together). In the case of `GpiCreateRegion`, a new region is created and its handle returned. In the case of `GpiSetRegion`, the previous definition is discarded and replaced by the new definition that is associated with the existing region handle.

Moving a Region

To move a region, call `GpiOffsetRegion`. This function accepts, as input, the region handle and an *offset* value that is added to every coordinate point of the region.

`GpiOffsetRegion` moves a region by a specific number of device coordinates. By specifying positive or negative x- and y-values an application can move a region in any direction, relative to its current position.

Determining Region Characteristics

Regions are used in many operations. There are several functions that determine the characteristics of regions because these characteristics influence the outcome of the operation being performed.

To determine whether 2 regions are identical, call `GpiEqualRegion`. This function accepts, as input, the 2 region handles. Regions are identical if the only difference between them is an empty region. For example, a rectangular region whose lower-left corner is at 10,10 is not identical to a region whose lower-left corner is at 50,50, even if the regions have exactly the same dimensions. The 2 regions also must be of the same device class to be considered identical.

To determine the rectangles that compose a region, call `GpiQueryRegionRects`, which retrieves the coordinates of a series of rectangles that, when ORed together, define the shape of the regions.

Note: The individual regions returned may differ in size and number from those originally used to specify the region

`GpiQueryRegionRects` accepts, as input, the region handle and the maximum number of rectangles that can be returned. The division of the region into multiple rectangles is handled automatically by the PM programming interface.

It is not necessary for an application to retrieve all the rectangles in a single function. Your application can call `GpiQueryRegionRects` any number of times. Therefore, the application must specify the maximum number of rectangles and the rectangle number to start from in each function.

No assumptions can be made about the number and coordinates of rectangles returned except that they will define the specified region precisely. The purpose of specifying the number of rectangles in the query is to ensure that the system does not return more coordinates than can fit in the `RECTL` array.

To determine whether a point is inside the borders of a region, call `GpiPtInRegion`. This function is especially useful in applications that must determine whether the mouse pointer is over a region. The particular point must be expressed in device coordinates.

To determine whether any part of a specified rectangle lies within a region, call `GpiRectInRegion`. This function accepts, as input, the region handle and the rectangle as specified in device coordinates.

To determine the coordinates of the smallest rectangle that encloses a region, call `GpiQueryRegionBox`. This function accepts the region handle as input. The output from this function tells you whether the bounding region is rectangular, `NULL`, or complex.

Converting a Region to a Clip Region

Clipping is the process an application uses to limit graphics output to a specific area (called the clipping area) of the display or page. Clipping is discussed in Chapter 16, "Clipping and Boundary Determination."

There are several clipping functions provided by the PM programming interface. However, if your application requires a clipping boundary in device coordinates, it must define the boundary with a region. To convert the region into a clipping boundary, call `GpiSetClipRegion`. The clip region, as defined by this operation, becomes the current clip region of all subsequent drawing operations.

`GpiSetClipRegion` accepts, as input, the region handle. A `NULL` region handle sets the clip region to infinity, effectively performing no clipping.

Unlike clip paths, the region that is no longer the current clip region is not deleted. It retains the effects of any changes made to it while it was a clip region; and it can be used with the other region functions, including being reselected as the clip region with `GpiSetClipRegion`.

You do not have to deselect the current clip region before selecting another. Each selected clip region automatically replaces the one before it. If there is an existing

clip region when you call `GpiSetClipRegion`, it reverts to a normal region, and its handle is returned.

Note: Clip regions and clip paths share the same low-level implementation. Use paths when you want to define your shape in world coordinates using drawing primitives, and regions when you want to define it in device coordinates (or world coordinates) using rectangles. Regions are generally faster to define.

When you have selected the current clip region, none of the region functions described thus far can be used for that region. The following functions can be used with the current clip region:

- `GpiQueryClipBox`
- `GpiIntersectClipRectangle`
- `GpiExcludeClipRectangle`
- `GpiOffsetClipRegion`
- `GpiPtVisible`
- `GpiRectVisible`.

These functions are described in Chapter 16, “Clipping and Boundary Determination.” All of these functions work in world coordinates, rather than device coordinates, and therefore, are subject to current transformations.

`GpiPtVisible` and `GpiRectVisible` do not apply exclusively to clip regions.

When the screen contents are altered (for example, when a window is sized), you have to be able to repair the part of the screen image affected by the change. Figure 11-4 illustrates the necessary region.

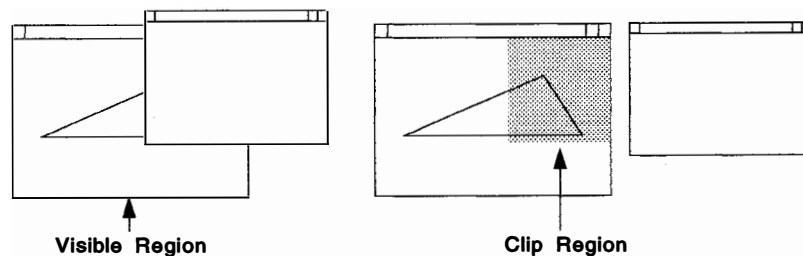


Figure 11-4. Repairing the Screen with Clip Regions

To improve performance of the drawing operation, you can restrict the redrawing and repair work to the affected parts of the screen.

Use `WinQueryUpdateRegion` to determine whether graphics objects are totally outside the update region and need not be drawn at all. Graphics objects that are within, or are partly outside, the update region should be drawn, and the system will perform the required clipping automatically.

Define a clipping region using the dimensions of the update region. Then call an appropriate GPI drawing request, such as `GpiDrawChain`, to redraw the screen contents. Any drawing that would occur outside the clip region is discarded according to the standard clipping rules. Only those graphics within the clip region are redrawn.

Deleting a Region

To delete a region, call `GpiDestroyRegion`. This function accepts, as input, the region handle. If the region is the current clip region, it cannot be deleted. To delete a current clip region, first call `GpiSetClipRegion` and specify a `NULL` region handle. The region is no longer the current clip region and can be deleted.

Using Regions

You can use the region functions to:

- Create or delete a region
- Combine regions
- Compare regions
- Move a region
- Paint and frame a region
- Locate a point with respect to a region
- Determine the coordinates of region rectangles.

Creating and Deleting a Region

To create a region:

1. Create an array of `RECTL` structures containing the dimensions of the rectangles that will compose the region.
2. Call `GpiCreateRegion` to create the region. (This function returns a handle that identifies the region.)

To delete a region, pass the handle returned by the `GpiCreateRegion` function to `GpiDestroyRegion`.

Figure 11-5 shows how to create and delete a region.

```
#include <os2.h>
void fncREGN01(void){
    HPS hps;                                /* Presentation-space handle */
    HRGN hrgn;                               /* Region handle */
    RECTL arcl[] = {
        25, 50,                               /* Rectangle 1 */
        75, 100,
        50, 75,                               /* Rectangle 2 */
        100, 150,
        75, 125,                               /* Rectangle 3 */
        200, 175,
        150, 75,                               /* Rectangle 4 */
        250, 150 };

    hrgn = GpiCreateRegion(hps,               /* Creates region */
        sizeof(arcl) / sizeof (RECTL), /* Number of rectangles in region */
        arcl);                               /* Array of rectangle structures */
    .
    . /* Work with the region here. */
    .
    GpiDestroyRegion(hps, hrgn); /* Destroys region identified by hrgn */
} /* fncREGN01 */
```

Figure 11-5. Creating and Deleting a Region

Combining Regions

To combine 2 regions:

1. Create a region that the operating system can use as the destination region when it combines the source regions.
2. Determine which of the 5 combining methods to use.
3. Call GpiCombineRegion.

Figure 11-6 shows how to combine 2 regions by using the OR operation:

```
#define INCL GPIREGIONS
#include <os2.h>
void fncREGN02(void){
    HPS hps;
    HRGN hrgn1, hrgn2, hrgn3;               /* Region handles */
    RECTL rcl1, rcl2;

    rcl1.xLeft = 50; rcl1.yBottom = 100;
    rcl1.xRight = 200; rcl1.yTop = 175;
    hrgn1 = GpiCreateRegion(hps, 1L, &rcl1); /* First source region */

    rcl2.xLeft = 125; rcl2.yBottom = 150;
    rcl2.xRight = 225; rcl2.yTop = 200;
    hrgn2 = GpiCreateRegion(hps, 1L, &rcl2); /* Second source region */

    hrgn3 = GpiCreateRegion(hps, 0L, (PRECTL) NULL); /* Destination region */

    /* Combine the regions. */
    GpiCombineRegion(hps, hrgn3, hrgn1, hrgn2, CRGN_OR);
} /* fncREGN02 */
```

Figure 11-6. Combining Regions

Comparing Regions

GpiEqualRegion determines whether two regions are identical. Figure 11-7 shows how to compare regions.

```
#define INCL_GPIREGIONS
#include <os2.h>
void fncREGN03(void){
    HPS hps;                /* Presentation-space handle */
    HRGN hrgn1, hrgn2;      /* Region handles */
    LONG lEqual;            /* Return value for GpiEqualRegion */
    RECTL rc11, rc12;       /* Structures for region coordinates */

    rc11.xLeft = 50; rc11.yBottom = 100;
    rc11.xRight = 200; rc11.yTop = 175;
    hrgn1 = GpiCreateRegion(hps, 1, &rc11); /* Creates first region */

    rc12.xLeft = 125; rc12.yBottom = 150;
    rc12.xRight = 225; rc12.yTop = 200;
    hrgn2 = GpiCreateRegion(hps, 1, &rc12); /* Creates second region */

    lEqual = GpiEqualRegion(hps, hrgn1, hrgn2); /* Compares regions */
    if (lEqual == EQRGN_EQUAL)
    {
        /* Regions are equal. */
    }
    else if (lEqual == EQRGN_NOTEQUAL)
    {
        /* Regions are not equal. */
    }
    else if (lEqual == EQRGN_ERROR)
    {
        /* An error occurred. */
    }
} /* fncREGN03 */
```

Figure 11-7. Comparing Regions

Offsetting a Region

GpiOffsetRegion moves a region, by a specified offset, from its current position (in world space). This function must be called with the address of a POINTL structure that contains an x and a y translation factor. Figure 11-8 shows how to offset a region.

```
} /* fncREGN04 */
POINTL pt1NewPos; /* Structure for offset value */

pt1NewPos.x = 200; /* Sets x offset */
pt1NewPos.y = 10; /* Sets y offset */

GpiOffsetRegion(hps, hrgn, &pt1NewPos); /* Offsets region */
```

Figure 11-8. Offsetting a Region

Painting a Region

GpiPaintRegion fills a region with the current fill pattern, using the colors and mix mode that appear in the current AREABUNDLE structure. GpiFrameRegion draws a frame around a region by tracing the perimeter of the region with a rectangle of a specified size. Figure 11-9 shows how to change the fill-pattern color to green, paint a region, and draw a frame around the region.

```
SIZEL sizl;                /* Structure for size of frame */
HPS hps;                   /* Presentation-space handle */
HRGN hrgn;                 /* Region handle */

GpiSetColor(hps, CLR_DARKPINK);
GpiPaintRegion(hps, hrgn); /* Paint region dark pink */
GpiSetColor(hps, CLR_BLACK);
sizl.cx = 5;
sizl.cy = 5;
GpiFrameRegion(hps, hrgn, &sizl); /* 5-by-5 black frame */
```

Figure 11-9. Painting a Region

Locating a Point with Respect to a Region

To determine whether the mouse pointer lies within a region:

1. Retrieve the mouse-pointer coordinates and store them in a POINTL structure.
2. Call GpiPtInRegion, passing it a handle that identifies the appropriate region and the address of the POINTL structure from Step 1.
3. Examine the value that GpiPtInRegion returns to determine whether the point lies within the region.

Figure 11-10 shows how to locate the mouse pointer with respect to a region.

```
#define INCL_GPIREGIONS
#include <os2.h>
void fncREGN04(void){
    POINTL ptlCurPos; /* Mouse coordinates */
    HPS hps;           /* Presentation-space handle */
    HRGN hrgn;         /* Region handle */
    LONG lPosition;    /* Return value for GpiPtInRegion */

    /* Determine mouse coordinates and store in ptlCurPos. */
    lPosition = GpiPtInRegion(hps, hrgn, &ptlCurPos);
    if (lPosition == PRGN_INSIDE) {
        /* Point lies within region. */
    } /* if */
    else if (lPosition == PRGN_OUTSIDE) {
        /* Point lies outside region. */
    } /* else-if */
}
```

Figure 11-10. Locating a Point in a Region

Determining Coordinates of Rectangles in a Region

If a region consists of more than one rectangle, you can call `GpiQueryRegionRects` to retrieve the coordinates of the lower-left and upper-right corners of each rectangle.

If you use `GpiQueryRegionRects` to retrieve every rectangle requested to define a region, the function retrieves the coordinates of as many contiguous rectangles as required. `GpiQueryRegionRects` returns the coordinates of the rectangles that define a region to an array of `RECTL` structures, and returns the number of rectangles that were requested for the definition to the **`crcReturned`** field of the `RGNRECT` structure. Your `RECTL` array may not be large enough to hold all of the rectangles. Specify the maximum it can accept and request the remainder in subsequent functions if necessary.

To determine the coordinates of the rectangles that form a region, follow these steps:

1. Create an array of `RECTL` structures that will receive the rectangle coordinates. More rectangles may be required to define the region than were required to create it.
2. Fill in the **`ircStart`**, **`crc`**, and **`usDirection`** fields of the `RGNRECT` structure. The **`crc`** field can specify more rectangles than will be returned by `GpiQueryRegionRects`.
3. Call `GpiQueryRegionRects` to retrieve the coordinates.

Figure 11-11 shows how to determine the number of rectangles that define a region, the coordinates of the rectangles in that region, and how to create a new region using those coordinates.

```
#define INCL_GPIREGIONS
#include <os2.h>
void fncREGN05(void){
    RGNRECT rgnrc;           /* Structure for region rectangles */
    RECTL arcl1[5];          /* Array for determining rectangle count */
    PRECTL parcl;            /* Array for rectangle coordinates */
    ULONG cRcts = 0;         /* Total number of rectangles in region */
    HPS hps;
    HRGN hrgn3;

    rgnrc.ircStart = 1;       /* Rectangle to start with */
    rgnrc.crc = 5;            /* Number of rectangles to query */
    rgnrc.ulDirection = RECTDIR_LFRT_BOTTOP; /* Direction to query */

    /******
    /* Determine the total number of rectangles in the region by
    /* repeatedly calling GpiQueryRegionRcts with an array of 5 RECTL
    /* structures. The loop terminates when the function retrieves less
    /* than 5 rectangles.
    /******
    do {
        GpiQueryRegionRcts(hps, /* Handle of presentation space */
            hrgn3, /* Region to query */
            (PRECTL) NULL, /* Gets all rectangles in region */
            &rgnrc, /* Structure with rectangle data */
            arcl1); /* Array of structures for coordinates */

        cRcts += rgnrc.crcReturned;
    } while (rgnrc.crcReturned == rgnrc.crc); /* While 5 rects retrieved */

    //cRcts = rgnrc.crcReturned + (rgnrc.ircStart - 1);

    rgnrc.ircStart = 0;       /* Rectangle to start with */
    rgnrc.crc = cRcts;        /* Number of rectangles to query */

    /* Allocate enough memory for all RECTL structures.
    parcl = (PRECTL) malloc(cRcts * sizeof(RECTL));

    /* Fill array with coordinates of all rectangles.
    GpiQueryRegionRcts(hps, /* Handle of presentation space */
        hrgn3, /* Region to query */
        (PRECTL) NULL, /* Gets all rectangles in region */
        &rgnrc, /* Structure with rectangle data */
        parcl); /* Array of structures for coordinates */
} /* fncREGN05 */
```

Figure 11-11. Determining Rectangle Coordinates in a Region

Summary

Table 11-1 summarizes the region object functions.

<i>Table 11-1. Region Functions</i>	
Function Name	Description
GpiCombineRegion	Combines 2 regions in any of 5 ways.
GpiCreateRegion	Creates a region from 1 or more rectangles. If there are 2 or more rectangles, they are combined with an OR operation.
GpiDestroyRegion	Deletes a region and frees the memory associated with it.
GpiEqualRegion	Determines whether 2 regions are equal.
GpiFrameRegion	Draws a frame around a region by tracing the perimeter.
GpiOffsetRegion	Moves a region, other than a clip region, in any direction by the specified amount.
GpiPaintRegion	Fills a region, other than a clip region, with the current fill pattern.
GpiPtInRegion	Determines if a point lies within a given region, other than a clip region.
GpiQueryRegionBox	Determines the dimensions of the smallest rectangle that surrounds the entire region.
GpiQueryRegionRects	Determines the coordinates of individual rectangles that compose a region.
GpiRectInRegion	Determines if all or part of a rectangle lies within a region.
GpiSetRegion	Replaces the definition of an existing region with a new definition described by one or more rectangles.

Table 11-2 summarizes the data structures used by the region object functions.

<i>Table 11-2. Region Structures</i>	
Structure Name	Description
RECTL	A structure specifying the values of 2 (x,y) coordinate pairs defining the bottom-left and top-right coordinates of a rectangle.
RGNRECT	A structure specifying a start rectangle number, a count of the maximum number of rectangles to be returned and a field to be updated with the number of region rectangles returned by GpiQueryRegionRects.

Chapter 12. Creating and Drawing Retained Graphics

There are two types of graphics output in the OS/2 operating system: *retained* and *nonretained*. This chapter describes creating and replaying retained graphics.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Coordinate spaces and transformations
- Editing graphic segments.

About Creating and Drawing Retained Graphics

An application draws by calling graphics functions. Applications store retained graphics in *segments*, which can be edited. This means that the retained image can be modified without having to re-create the unmodified portion using multiple GPI functions.

When using nonretained graphics, the output appears on the output device (for example, a window) immediately. However, if part of the picture is erased or must be repeated, the application must call the same graphics function a second, or even a third time.

There are many other advantages to using retained graphics, including:

- **Convenience**—An application can draw retained graphics with a single function that accesses the storage area containing all the individual functions necessary for the object.
- **Flexibility**—An application can redraw graphics retained in the segment store to any number of device contexts.
- **Editing**—An application can modify an object without having to call all the functions necessary to re-create the picture. Individual segments can be moved, scaled, or rotated; then, redrawn.

Editing the graphics within a segment is described in Chapter 13, "Editing Retained Graphics and Graphics Segments."

- **Power**—An application can access the more powerful and useful graphics functions only when graphics are stored in segments.
- **Organization**—An application can use segments to store the components that will be assembled into the final picture.

Primarily, a graphics segment is a means of grouping and storing graphics primitives and their attributes. Although the graphics within a segment usually are related in some way, they do not have to be. A segment might contain a number of unrelated GPI functions that you want to keep and execute at specific times.

Retained graphics do affect application performance, however, in that a normal presentation space requires 114KB of main memory more than a micro presentation space; and using the drawing mode `DM_RETAIN` adds an additional 46KB.

Notes:

1. Your application can have up to 16K (16378) segments in the segment store of a single presentation space. The size of an individual segment is limited only by the amount of storage available to you.
2. Do not confuse a graphic segment with a memory segment.

Drawing Modes

When you create a presentation space, the drawing mode is set to draw. Your application can change this mode using `GpiSetDrawingMode`. The two additional drawing modes provided by the PM programming interface are *retain* (`DM_RETAIN`) and *draw-and-retain* (`DM_DRAWANDRETAIN`). Your application can determine which drawing mode is set using `GpiQueryDrawingMode`.

The drawing mode that you select becomes current for the presentation space, and it can be changed any number of times. Select the appropriate drawing mode before creating a segment. The drawing mode effects the segment type.

Draw Mode

In *draw mode*, graphics output is provided immediately to the currently associated device and is not retained in a segment store. When the graphics have been drawn, they cannot be used again unless they are re-created. For example, when a window containing draw mode graphics is moved or sized, the graphics have to be re-created by the application.

Draw mode graphics are suitable if an application is creating fairly simple graphics quickly or is maintaining its own graphics database. In the latter case, there is nothing to gain by retaining graphics.

A segment created when the drawing mode is `DM_DRAW`, is called a *nonretained segment*. While it might sound contradictory, nonretained segments have certain advantages that are described in "Nonretained Graphic Segments" on page 12-11.

Retain Mode

In *retain mode*, graphics are retained in the segment store only; they are not directed to the current device as they are created.

In retain mode, the presentation space does not have to be associated with a device context when graphics are being defined. It is possible to define graphics and store their definitions without sending them to a display screen or printer. The concept of *attribute currentness*, however, is relevant only when you are drawing graphics to an output device. For example, the color, or other attribute, that is current when you define a primitive is the color in which the line is drawn. That color, or other attribute might *not* be the color that is current when you actually draw the primitive. This is described in "Attribute Currentness" on page 5-12.

Many of the GPI queries that return current attribute values are invalid in retain mode because current attribute values have no effect when graphics are not sent to an output device.

Draw-and-Retain Mode

In *draw-and-retain mode*, graphics are both drawn on the current device as they are created and stored for later use. The GPI queries that return current attribute values are valid in draw-and-retain mode.

Creating a Graphics Segment

Your application signals the start of a graphics segment using `GpiOpenSegment`, which is valid only in a normal presentation space. Figure 12-1 is an example of 2 segments in a presentation space.

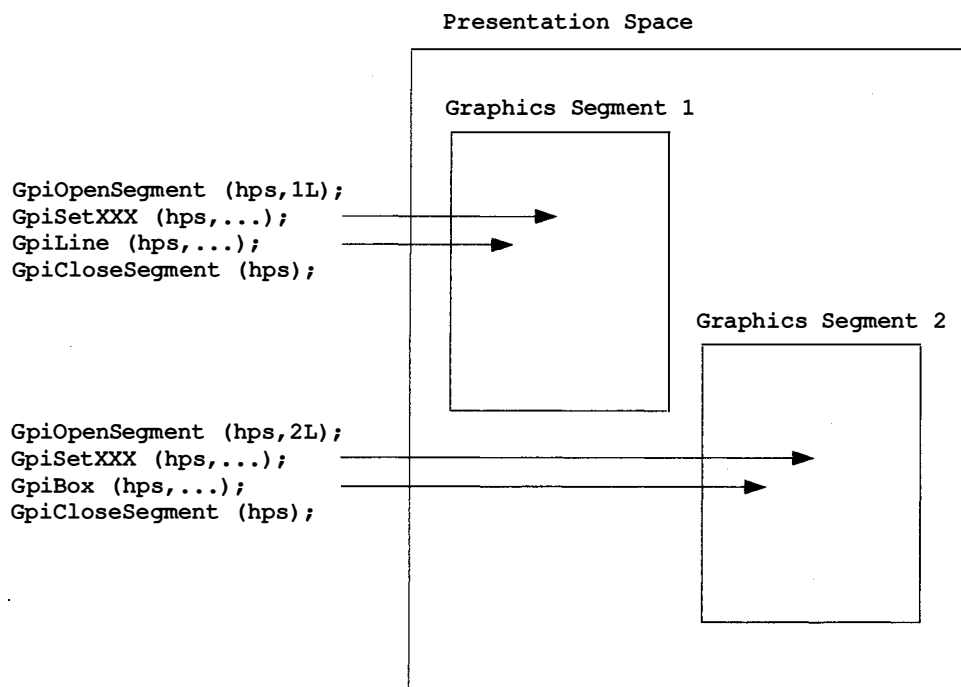


Figure 12-1. Graphics Segments in a Presentation Space. `GpiOpenSegment` accepts, as input, the presentation space handle and a long integer value, which names each segment.

OS/2 applications identify segments with long integer values. If you want to refer to the segment individually in later graphics operations (for example, to edit the segment) the identifier you supply must be greater than 0 and unique within the presentation space.

If you do not want to refer to the segment individually after it is created, you can give the segment an identifier of 0. You might do this when you are creating nonretained segments, for example. Any number of segments can have the 0 identifier; they are referred to throughout this guide as *zero segments*.

To determine which names already have been used in a presentation space, use `GpiQuerySegmentNames`. This function returns an array of segment names for all retained segments, excluding zero segments, within a specified name range.

Segments do not have to be named in consecutive order (although it is recommended for organizational purposes) because the segment order can be changed using `GpiSetSegmentPriority`.

Filling a Graphics Segment

After a graphic segment is opened, the GPI functions that follow become a part of a retained segment and are executed every time the segment, itself, is drawn.

The typical GPI functions that would be called are those that:

- Create graphics primitives—such as lines or markers
- Assign attributes to those primitives—such as color or line type.

However, there are a few GPI functions that you cannot call while there is an open segment, for example a second `GpiOpenSegment`. These functions are identified in Appendix B, “GPI Functions.”

A presentation space can contain many segments. Each time you open a new segment, many attributes reset themselves to default values. Therefore, the current position and attribute values that apply before you call `GpiOpenSegment` cannot be guaranteed to be in effect after you call the function. Beginning each segment with a number of attribute-setting requests and, possibly, with `GpiSetCurrentPosition`, is recommended. Your application also might take advantage of `GpiSetDefAttrs`, for example, to minimize the number of attributes that must be dealt with upon opening a new segment.

Closing a Graphics Segment

When you have finished creating a graphics segment, close it using `GpiCloseSegment`. There can be only one open segment at a time in a single graphics presentation space, so you must close one segment before going on to the next.

There is some degree of clean-up processing associated with using `GpiCloseSegment`, known as *close-segment processing*, that can make current attribute values unreliable. For more information about the effects of `GpiOpenSegment` and `GpiCloseSegment` on current attributes, see Appendix C, “Graphics Attributes.”

Segment Attributes

Each segment, whether retained or nonretained, has a number of characteristics, called *attributes* that you can set in accordance with your application's requirements. The attributes of a segment are quite different from those of a graphics primitive in that segment attributes have ON and OFF settings. There are 7 segment attributes; however, only 2 are described in this chapter: *chained* (`ATTR_CHAINED`) and *fast-chained* (`ATTR_FASTCHAIN`).

When an application creates a segment in a presentation space, the operating system assigns initial attributes to it. By default, 5 of the attributes will be set ON and 2 will be set OFF. The chained and fast-chained attributes are set ON. Your application can alter these values using `GpiSetInitialSegmentAttrs`, or retrieve the values of the current initial attributes using `GpiQueryInitialSegmentAttrs`.

The chain attribute tells the operating system to add each new segment in your application's presentation space to the segment chain. The fast-chained attribute tells the operating system to prevent the resetting of the primitive attributes to their default values before drawing the segment chain.

Chained Attribute

When you define a segment with the chained attribute switched ON, that segment becomes a part of the *segment chain* and is called a *chained segment*. The segment chain is composed of all chained segments defined in a single presentation space. Figure 12-2 on page 12-7 illustrates the segment chain concept. Segments can be chained together so that they can be drawn as a group. By default, each new segment is chained to the previous segment.

There can be only one segment chain at a time in a single presentation space, and all chained segments are chained to each other in the order in which you created them. Zero segments must have the chained attribute.

Usually, a logical relationship exists between the segments in a segment chain, although this is not a requirement. The whole segment chain can be drawn using `GpiDrawChain`. Each segment in the chain is called a *root segment*. Root segments are affected by those GPI functions that act on the segment chain, but they also can be manipulated independently of the chain.

Segments that are defined with the chained attribute switched OFF are called *unchained segments*. Your application can switch off the chain attribute using `GpiSetInitialSegmentAttrs` (hps, ATTR_CHAIN, ATTR_OFF). Unchained segments always are retained when they are created, regardless of the current drawing-mode parameter. An unchained segment must have a unique name.

There are a number of reasons for defining a segment as unchained. For example, a particular segment might not belong to the picture that is defined by the segment chain. You also are likely to define as unchained any segment that belongs to the picture but that has no single, fixed place in the segment chain. A car wheel, for example, could be defined once but drawn four times in a picture of a car. It would have no single, fixed place in the segment chain, but would be included four times in the picture by being *called* from one or more root segments.

A segment is called from another segment by including `GpiCallSegmentMatrix` in the calling segment. A segment and the segments it calls logically are one object. An important point about called segments is that they assume the primitive attribute settings of the calling segment. Of course, you can change the attribute settings within the called segment if the inherited values are inappropriate.

A closed, unchained segment can be called from any other segment. Chained segments, however, cannot be called from other segments.

Fast-Chained Attribute

The *fast-chained attribute* applies only to chained segments. It prevents primitive attributes from being reset to their default values at the beginning of the segment, an aid to performance. There is unnecessary overhead in resetting attributes to their defaults if either of the following is true:

- You are going to change the default values of the attributes at the start of the segment.
- You know that attributes have not been altered previously from their default values.

The fast-chained attribute is switched ON by default, and you should leave it on unless you specifically want attributes to be set to their default values. To turn off the fast-chained attribute, call `GpiSetInitialSegmentAttrs` (hps, ATTR_FASTCHAIN, ATTR_OFF).

Actual Drawing Mode

The drawing mode, as defined by `GpiSetDrawingMode`, works in conjunction with the segment status to produce the *actual drawing mode*. It is the actual drawing mode that determines whether graphics are:

- Drawn directly to the output device
- Retained in the segment store
- Both drawn and retained.

The actual drawing mode is summarized in Table 12-1.

Table 12-1. The Current Drawing Mode			
GpiSetDrawingMode parameter	Context		
	Chained Segment	Unchained Segment	Outside of any segment
DM_DRAWANDRETAIN	draw-and-retain	retain	draw
DM_RETAIN	retain	retain	draw
DM_DRAW	draw	retain	draw

As you can see in Table 12-1, graphics within chained segments always conform to the current drawing mode parameter. That is, they are drawn in `DM_DRAW` mode, retained in `DM_RETAIN` mode, and both drawn and retained in `DM_DRAWANDRETAIN` mode.

Graphics in unchained segments always are retained, regardless of the current drawing-mode parameter. You cannot retain segments created when the current drawing-mode parameter is `DM_DRAW`, unless they are unchained segments.

Similarly, graphics outside segments always are drawn without being retained. You cannot retain primitives outside segments, regardless of the current drawing mode.

Note: A micro presentation space has no segment store. Therefore, you cannot create graphics segments in a micro presentation space, nor can you change the drawing mode, which is always `DM_DRAW`.

Drawing a Retained Graphic

Your application can draw a complete segment chain with `GpiDrawChain`. The segments are drawn in the order in which they appear in the chain. For example, the segments in Figure 12-2 on page 12-7 are drawn in the following order:

1. Root segment 1
2. Unchained segment A
3. Unchained segment B
4. Root segment 2
5. Unchained segment B
6. Root segment 3.

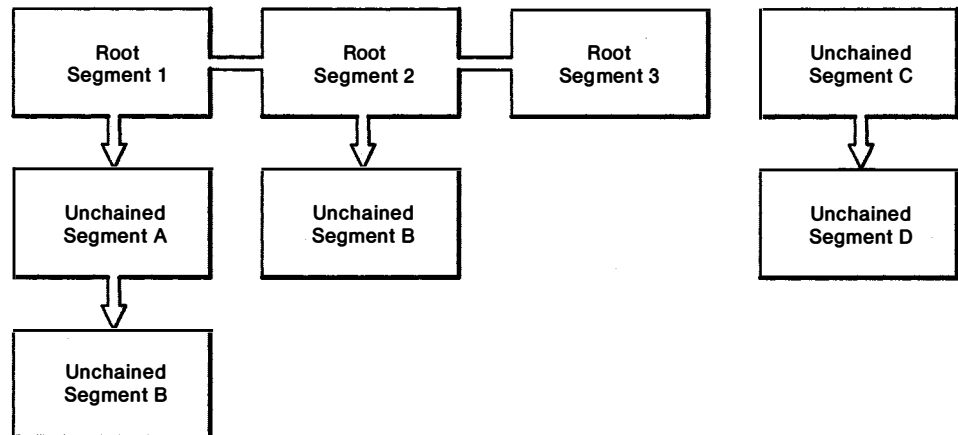


Figure 12-2. Chained and Called Segments. Segment A is called by root segment 1, and segment B is called by both segment A and root segment 2. Segment C calls segment D. Both C and D are unchained segments that are not called from the segment chain, and consequently, are not part of the segment chain.

`GpiSetSegmentPriority` changes the position of a root segment (which must not be a zero segment) in the chain. As input to this function, you supply the name of the segment you want to reorder and the name of a reference segment. The *reference segment* is the segment that either will be immediately before, or immediately after, the reordered segment's new position in the chain.

If the segment you are moving is to come after the reference segment in the chain, it is said to have a *higher priority* (`HIGHER_PRI`). If it is to come before the reference segment, it is said to have a *lower priority* (`LOWER_PRI`). The nearer a segment is to the end of the chain, the higher its priority.

If you supply the name of an unchained segment as input to `GpiSetSegmentPriority`, the segment is added to the chain in the position you specify. To learn the position of a segment in the chain, use `GpiQuerySegmentPriority`.

If your application called the following functions:

```

GpiSetSegmentPriority (hps, C, 2, LOWER_PRI)
GpiSetSegmentPriority (hps, 3, 0, HIGHER_PRI)
GpiSetSegmentPriority (hps, B, 0, LOWER_PRI)

```

the segment chain in Figure 12-2 would appear as in Figure 12-3 on page 12-8; and `GpiDrawChain` would draw the segments in the following order.

1. Root segment 3
2. Root segment 1
3. Unchained segment A
4. Unchained segment B
5. Segment C
6. Unchained segment D.
7. Root segment 2
8. Unchained segment B.
9. Segment B

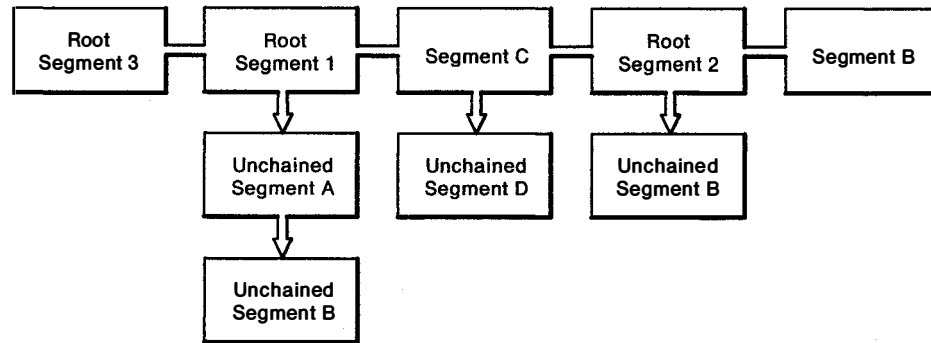


Figure 12-3. Chained and Called Segments Reordered Using *GpiSetSegmentPriority*

Segment Priority

The priority of a segment, in conjunction with the current foreground and background mix attributes, affects how the picture is presented to the user. The segment with the highest priority is drawn last, and appears on top of the previously drawn primitives. Picture components in segments with low priorities risk being drawn over and never seen by the user. The mix attributes affect the graphics functions within each segment.

GpiDrawSegment

In Figure 12-2 on page 12-7, Unchained Segment C and its called segment Unchained Segment D are not part of the segment chain and, therefore, are not drawn by *GpiDrawChain*. Both segments can be drawn using *GpiDrawSegment* (hps, C).

The function *GpiDrawSegment* accepts as input any segment name greater than zero. *GpiDrawSegment* can draw individual segments both inside and outside the segment chain. Any segment that is called from the *GpiDrawSegment*-named segment also is drawn.

GpiDrawFrom

You can draw a smaller portion of the entire segment chain (but a larger portion than a single segment) using *GpiDrawFrom*. *GpiDrawFrom* accepts, as input, a presentation space handle and 2 nonzero segment names, the first of which must be part of the segment chain. *GpiDrawFrom* then draws the segments, starting with the first and ending with the last, including all chained and called segments. The order in which the segments are drawn is identical to that of Figure 12-2 on page 12-7—a root segment, its first-to-last called segments, then the next root segment.

If the second named segment does not appear between the first segment and the end of the chain, the drawing begins with the first and continues through the end of the chain, including all chained and called segments, again with the drawing order as described in Figure 12-2 on page 12-7.

Attribute Modes

You can change primitive-attribute settings at any time; the new values you specify replace the existing values. For example, if the current foreground color is green and you change it to red, red replaces green as the foreground color attribute.

When you set primitive attributes within a retained or nonretained segment, you can save the existing attribute values on a last-in-first-out (LIFO) stack, from where they can later be retrieved and made current again. You do this by selecting either of the 2 attribute modes:

- **AM_PRESERVE mode**—Also known as *push-and-set mode*
- **AM_NOPRESERVE mode**—Also known as *set mode*.

In **AM_NOPRESERVE** mode, which is the default setting, existing attribute values are replaced by any new attribute values that you supply. In **AM_PRESERVE** mode, the existing attribute values are stored on a LIFO stack; then, the new values that you specify take effect. **AM_PRESERVE** mode also causes the current position to be saved if the position was specified using `GpiSetCurrentPosition`. If you use `GpiMove` to set the current position, the position is not saved, regardless of the attribute mode. Those attributes that are affected by the current attribute mode are listed in the *Presentation Manager Programming Reference*.

To select a current attribute mode, use `GpiSetAttrMode`. The current attribute mode affects subsequent attribute-setting requests in the presentation space. Attribute modes are not applicable to micro presentation spaces, because graphics segments can be created only in normal presentation spaces.

To retrieve an attribute value from the LIFO stack, use `GpiPop`, which pulls back the attribute that was stored most recently on the stack. You can retrieve more than one attribute value by supplying a number as an input parameter of this function. For example, if you specify 4, `GpiPop` retrieves the 4 attributes that were most recently stored on the stack. If each of the values retrieved applies the same type of attribute to the same primitive (for example, if all four are line-type settings), the last one to be retrieved (and, therefore, the first one on the stack) becomes the current setting.

If you save attribute values from any segment called from another segment, and do not retrieve the values using `GpiPop`, the values are restored automatically when the end of the segment is reached. If you save attribute values from any segment that is not called from another segment, and do not explicitly restore those values using `GpiPop`, they are lost at the end of the segment.

The **AM_PRESERVE** mode is useful when you do not want attributes in calling segments to be overwritten by attributes specified in the called segments. This overwriting happens because a calling segment and the segments it calls are logically one object. Attribute changes within a called segment remain current upon return to the calling segment. If you set some attribute values at the start of a called segment, and the current attribute mode is **AM_PRESERVE**, the attribute values of the calling segment are stored on the LIFO stack. At the end of the called segment, the values on the stack are retrieved automatically so that the calling segment continues with its own attribute values.

Reusing the Presentation Space

A normal presentation space with segments retained in the segment store can be costly in terms of storage. Therefore, delete any presentation space that you no longer need, using `GpiDestroyPS`. If for example, your application is using a series of presentation spaces, each with a different format, the creation and deletion activity also can be costly. The PM programming interface provides the following functions that let you reuse or redefine an existing presentation space; so you can create a presentation space once, and use it any number of times.

- `GpiSavePS`
- `GpiRestorePS`
- `GpiResetPS`
- `GpiSetPS`.

GpiSavePS

`GpiSavePS` saves presentation space information (such as current primitive attributes and the current position) on a dedicated LIFO stack. The presentation space itself is unchanged; that is, it still exists, has the same presentation space handle, and the presentation page dimensions and format are the same.

Output from `GpiSavePS` is a number that identifies the saved information on the LIFO stack. An output of 3, for example, tells you that this is the third lot of presentation space data on the stack.

GpiRestorePS

`GpiRestorePS` retrieves the saved information from the LIFO stack and reapplies it to the presentation space. Input to this function can be either the identifier returned to you from `GpiSavePS` or a relative value. A relative value of `-1`, for example, retrieves the information most recently stored on the stack. You do not have to restore the information that was most recently saved; however, any data that you skip over is discarded.

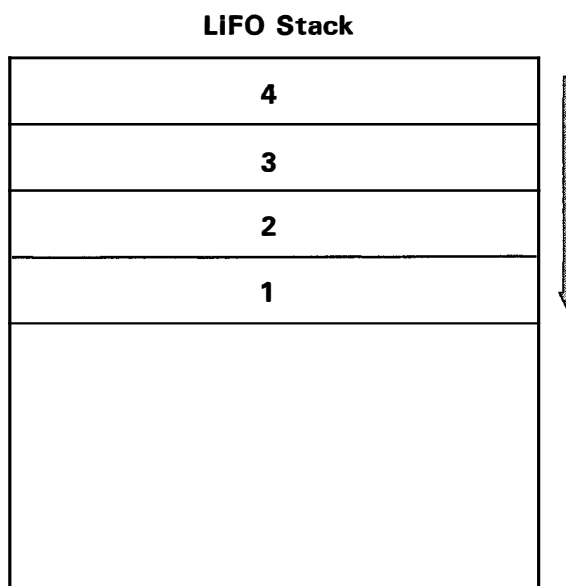


Figure 12-4. A LIFO Stack with Four Items

For example, if there are four items of data on the stack, as shown in Figure 12-4, and you request the restoration of item 2, items 3 and 4 are discarded. Item 1 is the only item remaining on the stack.

You can restore item 2 by specifying either an absolute value of 2, or a relative value of -3. The relative value depends on the number of items currently in the stack. The contents of the LIFO stack are purged whenever GpiAssociate is called.

GpiResetPS

When a presentation space is first created, it is in a neutral state. Current attributes are set to their initial default values; the current position is (0,0); the segment store contains no segments; and, there are no application-defined resources, such as logical color tables. You can return an existing presentation space to this state using GpiResetPS. GpiResetPS has 3 levels of reset activity, each more powerful than the last. These are:

1. GRES_ATTRS, which is equivalent to the processing done by GpiCloseSegment.
2. GRES_SEGMENTS, which is equivalent to creating a new presentation space, but without deleting any logical resources. All retained segments are deleted from the segment store.
3. GRES_ALL, which is equivalent to creating a new presentation space.

GpiResetPS does not alter the size or format of the presentation page.

GpiSetPS

GpiSetPS redefines the size and format of the presentation page. The processing performed when you call GpiSetPS is similar to that performed by GpiCreatePS, except that the presentation space already exists. The presentation space is returned to a neutral state (which is the equivalent of requesting GRES_ALL using GpiResetPS), and the presentation page is redefined. For example, you can change a presentation page defined in 0.01mm units to one defined in pels. Essentially, this lets you work with a new presentation space without having to delete one and create another.

You also can use GpiSetPS to change the current mapping mode of a presentation space. To do this, use the PS_NORESET flag, which is the equivalent of requesting GRES_SEGMENTS using GpiResetPS. This feature is particularly useful if you are designing an application that deals with page layout and drafting, or if you want the screen size to correspond to the page size for the printed output.

Nonretained Graphic Segments

It is valid segment construction to create a segment bracket while the drawing mode is DM_DRAW. The GPI functions contained within the bracket are drawn immediately rather than being retained for future use; thus, this type of segment is called a *nonretained segment*.

Because the usual reason for constructing segments is to take advantage of retained graphics, nonretained segments might appear as a contradiction in terms at first. However, there are four particular advantages in their use:

- If ATTR_FASTCHAIN is set to OFF, the GpiOpenSegment implicitly resets all attributes to their defaults.
- The GpiOpenSegment and GpiCloseSegment initialize and reset the viewing transform matrix, just as they do for retained segments.
- Nonretained segments can be recorded in a metafile, just as a retained segment can.
- A graphic in a nonretained segment, with a unique name, can be converted easily to a retained graphic for future use.

Using Segment Creating and Drawing Functions

You can use retained-drawing and segment functions to:

- Create a chained or called segment
- Draw the picture associated with one or more segments.

Creating a Chained Segment

To create a chained segment, you must:

1. Set the drawing mode to DM_RETAIN.
2. Check to see if the chained attribute is one of the initial segment attributes using GpiQueryInitialSegmentAttrs.
3. Set the chained attribute, if necessary, with GpiSetInitialSegmentAttrs.
4. Open the segment using GpiOpenSegment.
5. Perform the necessary drawing operations.
6. Close the segment using GpiCloseSegment.

Figure 12-5 is an example of a segment containing a box primitive and calling another segment using GpiCallSegmentMatrix.

```
#define INCL_GPISEGMENTS
#define INCL_GPITRANSFORMS
#include <os2.h>
void fncSEGS01(void){
    POINTL ptl;
    HPS hps;
    LONG idSegment = 1;
    LONG idNonChained = 2;
    MATRIXLF matlfrTransform = { MAKEFIXED(2,0), MAKEFIXED(0,0), 0,
                                MAKEFIXED(0,0), MAKEFIXED(1,0), 0,
                                0, 0, 1 };

    /******
    /* Turns chaining on. Adds the new segment to the segment chain.          */
    /* Segment idNonChained is called, whether chained or not.                */
    /******

    if (ATTR_OFF == GpiQueryInitialSegmentAttrs(hps, ATTR_CHAINED)
        GpiSetInitialSegmentAttrs(hps, ATTR_CHAINED, ATTR_ON);
    GpiOpenSegment(hps, idSegment);
    ptl.x = 150; ptl.y = 150;
    GpiMove(hps, &ptl);
    ptl.x = 225; ptl.y = 225;
    GpiBox(hps, DRO_FILL, &ptl, 0L, 0L);
    GpiCallSegmentMatrix(hps, idNonChained, 9L, &matlfrTransform,
        TRANSFORM_REPLACE);
    GpiCloseSegment(hps);
} /* fncSEGS01 */
```

Figure 12-5. Creating a Chained Graphics Segment

Creating a Called Segment

To create a called segment, you must:

1. Set the drawing mode to DM_RETAIN.
2. Check to see whether the chained attribute is one of the initial segment attributes using GpiQueryInitialSegmentAttrs.
3. Set the chained attribute, if necessary, with GpiSetInitialSegmentAttrs.
4. Open the segment using GpiOpenSegment.
5. Perform the necessary drawing operations.
6. Close the segment using GpiCloseSegment.

Figure 12-6 shows an example of how to draw a box in a called segment.

```
#define INCL_GPISEGMENTS
#define INCL_GPICONTROL
#include <os2.h>
void fncSEGS02(void){
    POINTL ptl;
    HPS hps;
    LONG idNonChained = 2;

    GpiSetDrawingMode(hps, DM_RETAIN);

    /* Creates a non-chained segment. */

    if (ATTR_ON == GpiQueryInitialSegmentAttrs(hps, ATTR_CHAINED))
        GpiSetInitialSegmentAttrs(hps, ATTR_CHAINED, ATTR_OFF);
    GpiOpenSegment(hps, idNonChained);
    ptl.x = 100;
    ptl.y = 100;
    GpiMove(hps, &ptl);
    ptl.x = 200;
    ptl.y = 200;
    GpiLine(hps, &ptl);
    GpiCloseSegment(hps);
} /* fncSEGS02 */
```

Figure 12-6. Creating a Called Graphics Segment

Drawing a Segment Chain

Figure 12-7 shows an example of how to draw a segment chain using the GpiDrawChain function.

```
#define INCL_GPICONTROL
#include <os2.h>
void fncSEGS03(void){
    HPS hps;

    if (DM_DRAW != GpiQueryDrawingMode(hps))
        GpiSetDrawingMode(hps, DM_DRAW);
    GpiDrawChain(hps);
} /* fncSEGS03 */
```

Figure 12-7. Drawing a Segment Chain

Summary

Table 12-2 summarizes the functions used to create and draw retained graphics and segments.

<i>Table 12-2. Retained Graphics and Segment Functions</i>	
Function Name	Description
GpiCallSegmentMatrix	Concatenates the contents of an unchained segment bracket onto another chained or unchained segment bracket.
GpiCloseSegment	Defines the end of a segment bracket.
GpiDrawChain	Draws the subpictures stored in a presentation space segment chain.
GpiDrawFrom	Draws the subpictures from a specified range of segments in the segment chain.
GpiDrawSegment	Draws a subpicture from a single segment.
GpiOpenSegment	Defines the beginning of a segment bracket.
GpiQueryInitialSegmentAttrs	Retrieves the setting of 1 of the 7 initial segment attributes.
GpiQuerySegmentAttrs	Retrieves the setting of 1 of the 7 segment attributes.
GpiQuerySegmentNames	Retrieves a list of existing segment identifiers within a specified range.
GpiQuerySegmentPriority	Returns the identifier for a segment that precedes or follows a segment in the segment chain.
GpiSetDrawingMode	Sets the drawing mode in your presentation space to 1 of 3 possible modes: draw, retain, or draw-and-retain.
GpiSetInitialSegmentAttrs	Sets the default segment attributes.
GpiSetSegmentAttrs	Sets 1 of the 7 segment attributes.
GpiSetSegmentPriority	Alters the order in which the operating system draws and detects segments in the segment chain.

Chapter 13. Editing Retained Graphics and Graphics Segments

This chapter describes editing retained graphics and graphics segments. The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Coordinate spaces
- Transformations
- Creating segments.

About Editing Retained Graphics and Graphics Segments

In the OS/2 operating system, applications store retained graphics in *segments*. One of the advantages of using graphics segments is that segments can be edited, which allows the retained image to be modified without having to re-create the unmodified portion with multiple GPI functions.

Chapter 12 described how to create a segment, and store GPI functions within the segment bracket. To understand how to edit a segment, the underlying structure of a segment is described here in greater detail.

The GPI functions are not inserted directly into a segment bracket. Instead, the operating system converts the GPI functions into graphics orders; then stores these orders in the GpiOpenSegment—GpiCloseSegment bracket.

Generally, each of the GPI functions within the segment bracket generates one *element* of the segment. An element is the smallest portion of a segment that can be edited and is made up of one or more orders. A graphics order, also known as a drawing order, is the smallest, complete portion of a segment. Figure 13-1 illustrates the levels of graphic segment construction.

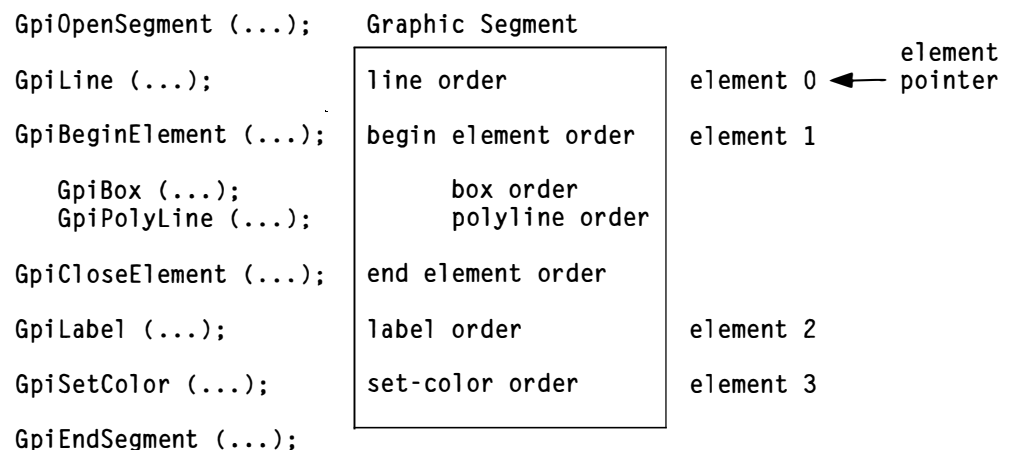


Figure 13-1. Structure of a Graphics Segment. Elements 0, 2, and 3 are each composed of a single order. Element 1 is composed of two orders bracketed by GpiBeginElement and GpiEndElement.

Graphics Orders

A graphics order is a low-level graphics command that corresponds to a primitive function or attribute. In addition to code and data requirements, each graphics order uses approximately 11 bytes of storage. An application that uses 2000 graphics drawing orders will use around 22KB of memory to store them. Table 13-1 describes the four sizes of graphics orders.

Table 13-1. Graphics Orders	
Graphics Order Size	Content
1 byte	A hexadecimal identifier corresponding to a drawing function or attribute function. This identifier is also known as the order code.
2 byte	The order code is in the first byte, and data is in the second byte.
Long	• The order code is in the first byte. • The length value of the actual data, in bytes, is in the second byte. • The actual data (up to 255 bytes).
Very long (maximum length of 64KB)	• A hexadecimal identifier specifically for extended orders, is in the first byte. • The order code is in the second byte. • A length value that specifies how many bytes are used by the graphics-order arguments, (high order) is in the third byte. • A length value that specifies how many bytes are used by the graphics-order arguments, (low order) is in the fourth byte. • The actual data.

The following example shows a long graphics order that corresponds to the GpiLine function:

81 8 100 0 0 0 100 0 0 0

The first number, 81, is the hexadecimal identifier that corresponds to GpiLine. The second number, 8, is the length value that specifies how many bytes are used by the graphics-order arguments. The next 8 bytes contain the arguments for GpiLine. In this case, the arguments specify the line's end point at (100,100).

In most cases, graphics orders in a segment correspond to a subpicture, which is part of a complete, more complex picture. Your application would combine the individual segments to form the complete picture.

Three drawing modes affect how the operating system stores graphics orders in segments. These modes are described in Table 13-2 on page 13-3. The default drawing mode is draw. To specify another as the current drawing mode, use GpiSetDrawingMode.

The *actual drawing mode* is a combination of the drawing mode as set using GpiSetDrawingMode, and the segment status—chained, unchained, or outside of the segment. Table 12-1 on page 12-6 describes the actual drawing mode. The actual drawing mode does not effect the storing of orders in segments.

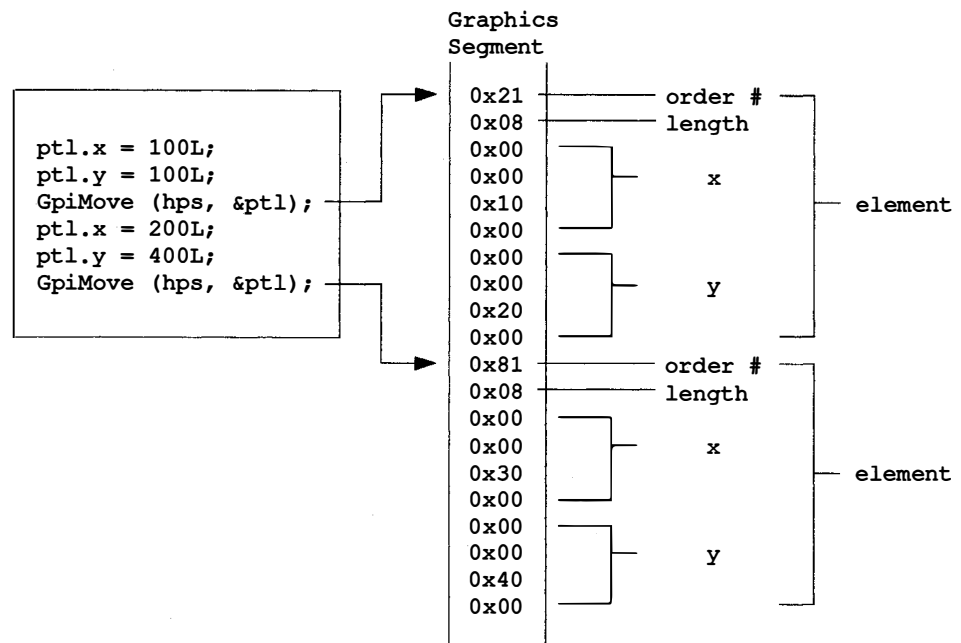


Figure 13-2. Graphics Orders. The encoding that appears in the Graphics Segments column is a hexadecimal version of the order, length, and parameter information. The GPIs and their corresponding graphics orders are listed in *Presentation Manager Programming Reference*.

Table 13-2. Segment Graphics Drawing Modes		
Drawing Mode	GpiSetDrawingMode Value	When this mode is set...
Draw	DM_DRAW	it is not possible to store graphics orders in a chained segment.
Retain	DM_RETAIN	your application can store graphics orders in chained and unchained segments.
Draw-and-retain	DM_DRAWANDRETAIN	your application can store graphics orders in chained and unchained segments. In this mode, output intended for a chained segment is both drawn on the device and stored in a segment.

Graphics Elements

Graphics elements are composed of either a single graphics order, or a series of graphic orders that are bracketed by an opening and closing element function.

The purpose of this element bracket is to allow the addition of a single element, that comprises more than one graphics order, to a segment. This is most useful when you know that you will want to retrieve those orders at a later time, and manipulate them as a group rather than as individual functions.

Note: There is no reason for enclosing a single GPI function within a GpiBeginElement—GpiEndElement bracket, although it is a valid construction.

Element brackets also are valid when drawing graphics directly to the output device, but again, they serve no purpose unless converted to a retained segment at a later time.

Elements cannot be nested, so two elements cannot begin in succession without an intervening `GpiEndElement` request. Certain GPI functions cannot be called between `GpiBeginElement` and `GpiEndElement`.

Element construction is valid only within segments.

Adding Elements to a New Segment

There are four ways of adding elements to a newly created segment:

- Call one or more GPI functions. Each function generates one element of the segment. This is the simplest way of supplying graphics data to a segment.
- Call `GpiBeginElement`, and follow it with one or more GPI functions. Close the element using `GpiEndElement`. All the GPI functions enclosed within the bracket, and the `GpiBeginElement`, and `GpiEndElement` functions generate a single element of the segment, and therefore occupy a single element-pointer position. Subsequently, the graphics orders cannot be accessed individually with the element pointer, although the orders can be accessed directly, for example, using `GpiGetData`.
- Call `GpiElement`. As input, you supply the graphics orders themselves, rather than the GPI functions that generate them. No matter how much data you supply on this request, it is considered a single element of the segment.

Multiple orders added to a segment in this way cannot be accessed individually using the element pointer, but they can be accessed directly, for example, using `GpiGetData`.

`GpiElement` cannot be included within a `GpiBeginElement`—`GpiEndElement` bracket. The data passed by `GpiElement` must not include `GpiBeginElement` and `GpiEndElement`.

- Call `GpiPutData`. As with `GpiElement`, supply graphics orders as input. You can even include `GpiBeginElement` and `GpiEndElement` orders in this data, and thus add more than one element to the segment. `GpiPutData` is useful when there is a large amount of data to add to a segment.

`GpiPutData` cannot be used if the segment editing mode is set to `SEGEM_REPLACE`. When the segment editing mode is `SEGEM_INSERT`, the element pointer must point to the last element in the current segment.

`GpiPutData` is most often used in conjunction with `GpiGetData`, which copies data as a list of drawing orders from a segment to application storage.

Element Types

Each element that results from a single GPI function has an associated *element type*.

The element type is generally the graphics-order code associated with the graphics order generated by the GPI function, and is supplied by the system. For example, `GpiLine` has the element type, `OCODE_GCLINE`, which identifies it as a line-drawing request. Your application can determine the current classification of an element using `GpiQueryElementType`.

The specific element type for each of the GPI functions are documented in the *Presentation Manager Programming Reference*. These types are equivalent to the graphic order values.

When using `GpiElement` or `GpiBeginElement`, you can simultaneously add several graphics orders to a segment. Those orders become one element of the segment. You can supply your own classification for these compound elements by supplying a value for the type parameter of `GpiElement` or `GpiBeginElement` in the range of 0x81000000 through 0xFFFFFFFF.

For example, the orders passed by a single `GpiElement` function could change the line type to dotted, set the current color to blue, and draw a line. You could classify this element with a type value that identifies it as a blue dotted line.

Note: Element types are used only for classification. They are not used as location markers or correlation tags.

The Element Pointer

Use the *element pointer* to change the contents of an existing segment. When calling `GpiOpenSegment`, the element pointer is set to 0. The first element that you add to a newly created segment causes the pointer to be set to 1, and each subsequent element causes the pointer to be incremented by 1. When a segment is closed, the element pointer is always reset to 0.

Your application can move the element pointer to a specified value using `GpiSetElementPointer`. For example, to address the sixth element in a segment, set the element pointer to 6.

Your application also can move the element pointer by a specific number, rather than to a specific number, using `GpiOffsetElementPointer`. For example, if the element pointer currently addresses element 3, to move it to element 8, code an offset of 5 in `GpiOffsetElementPointer`. The pointer moves backward rather than forward through the segment. The offset value can be a positive or negative number.

If the sum of the offset and the current pointer position is less than 0, the operating system sets the pointer so that it points to position 0. Position 0 precedes the first element in the segment. If the sum of the offset and the current pointer position is greater than the number of elements in the segment, the operating system sets the pointer so that it points to the last element in the segment.

You can determine the current location of the element pointer using `GpiQueryElementPointer`. To request the contents of a part of or all of the elements at the current pointer position, use `GpiQueryElement`.

Labels

You can include *labels* in a segment to make locating particular elements of the segment easier. If you label an element that is likely to be edited, you can access that element when necessary, regardless of whether the actual position of the label – element pair within the segment has changed.

Warning: When editing segments, do not inadvertently insert elements between a label and the element that follows. Your application will no longer be able to use certain GPIs correctly. Their default locations and offsets will be invalid.

A label is only a place-holder or reference point, and is itself an element of the segment. Labels are long integer values. To include a label in a segment, use

GpiLabel. The following example includes the label 5 in the current segment, whose segment id is 4.

```
GpiOpenSegment(hps, 4L);
GpiLabel(hps, 5L);
:
```

Your application can set the element pointer to the label position using GpiSetElementPointerAtLabel, then increment the pointer position to the element itself using GpiOffsetElementPointer with an offset of 1.

Note: This example of GPI cannot work properly if your application has allowed the insertion of elements between a label and the element that follows it.

Labels do not have to be unique. If you do not use unique labels, GpiSetElementPointerAtLabel positions the element pointer at the first occurrence of the label, starting from the current pointer position. If the label is not found between the current pointer position and the end of the segment, an error condition is raised. Using labels as a navigation aid might be quicker than scanning the segment for a particular element.

If you do not use labels, you must locate an element before you can change it. To locate a specific element in a segment, you must repeatedly move the element pointer using GpiSetElementPointer, then read the contents of the element using GpiQueryElement. This can be quite a lengthy procedure depending upon the number of elements in the segment.

Graphics Segments

Segments are made up of elements, which are in turn made up of one or more graphic orders. The previous sections have discussed the properties and functions for orders and elements. The following sections discuss the attributes and functions that apply to segments as a whole.

The segment attributes are encoded in a segment prior to the graphics orders that correspond to GPIs and attributes.

Initial Segment Attributes

Each segment has a number of characteristics, called *attributes* that you can set in accordance with your application's requirements. The attributes of a segment are quite different from those of a graphics primitive, in that segment attributes have ON and OFF settings. There are seven segment attributes, each described in Table 13-3.

Table 13-3 (Page 1 of 2). Graphic Segment Attributes and Initial Settings			
Attribute	Value	If set...	Default
Chained	ATTR_CHAINED	the operating system adds each new segment in your application's presentation space to the segment chain.	ON
Fast chain	ATTR_FASTCHAIN	the operating system prevents the resetting of primitive attributes to their default values before drawing the segment chain.	ON

Table 13-3 (Page 2 of 2). Graphic Segment Attributes and Initial Settings

Attribute	Value	If set...	Default
Dynamic	ATTR_DYNAMIC	the operating system draws segment output by using the XOR raster operation.	OFF
Detectable	ATTR_DETECTABLE	your application can perform correlation operations on segments created in this presentation space.	OFF
Propagate detectable	ATTR_PROP_DETECTABLE	the detectable attribute will be set in each segment called by a chained segment.	ON
Visible	ATTR_VISIBLE	GpiDrawChain, GpiDrawFrom, and GpiDrawSegment will generate output on a device.	ON
Propagate visible	ATTR_PROP_VISIBLE	The visible attribute will be set in each segment called by a chained segment.	ON

When an application creates a segment in a presentation space, the operating system assigns initial attributes to it. By default, five of the attributes will be set ON and two will be set OFF. Your application can override the defaults for all subsequently created segments within the presentation space using `GpiSetInitialSegmentAttrs`.

Use `GpiQueryInitialSegmentAttrs`, to retrieve the values of the current initial attributes.

Note: The chained and fast-chained attributes are discussed in “Chained Attribute” on page 12-5 and “Fast-Chained Attribute” on page 12-5.

The Dynamic Attribute

Dynamic segments are graphics segments that can be moved from where they were drawn on the screen to a different coordinate position, or altered in some other way, without affecting the remaining part of the picture. The new position of a dynamic segment can be provided with an input device, such as a mouse, or it can be application-generated.

The dynamic attribute is set to OFF by default. The setting is always inherited by called segments from the setting of their callers. Thus, if a calling segment is nondynamic, any segments called by it are also nondynamic, regardless of how they have been defined. If the calling segment is dynamic, all segments called by it are also dynamic. Any segment explicitly defined as dynamic must have a unique name, and it cannot be created when the current drawing-mode parameter is `DM_DRAW` or `DM_DRAWANDRETAIN`. Although no error condition is raised if you define an unchained segment as dynamic, it is not treated as dynamic unless the segment is called from a dynamic root segment.

Graphics objects to be moved without disturbing the remaining contents of the display are drawn in exclusive-OR mode. Segments that you define as dynamic always are drawn in exclusive-OR mode, regardless of the current mix attribute settings and any GpiSetMix functions the segment itself might contain.

For performance reasons, dynamic segments are to be grouped at the start of the picture chain. There are GPI functions that handle dynamic segments as a group, so it is more efficient if the entire segment chain does not have to be scanned to locate them.

When the entire picture chain is drawn, however, dynamic segments are to be drawn *after* all other segments in the chain to ensure that the effects of drawing in exclusive-OR mode are not adversely affected by drawings in other mix modes. Also, you must ensure that no nondynamic drawing overlays the dynamic segments after they have been drawn. The PM programming interface ensures that dynamic segments are always drawn on top of other graphics.

The Detectability Attribute

The detectability attribute of a segment determines if that segment can be selected with an input device, such as a mouse. A segment with this attribute is said to be *selectable*. Selectable segments can be selected for correlation operations. The detectability attribute is set OFF by default. All detectable segments must have unique names. If you define a zero segment as detectable, it is created as nondetectable.

The Propagate-Detectability Attribute

This attribute controls whether the detectability attribute of a calling segment is inherited by any segments called from it. The propagate-detectability attribute is set ON by default. This value overrides the detectability setting of the called segment. For example, if you set the detectability attribute of a calling segment to OFF, all segments called from it are nondetectable, regardless of how they have been defined.

The Visibility Attribute

The visibility attribute controls whether a segment is to be visible on an output device. The contents of a segment that is not defined with the visibility attribute set ON are still executed whenever the segment is drawn. This can cause changes to the current position and to current attribute values, even though the primitives themselves are not visible on the output device. The visibility attribute is set ON by default.

The Propagate-Visibility Attribute

This attribute controls whether the visibility attribute of a calling segment is inherited by any segments called from it. The propagate-visibility attribute is set ON by default. This value overrides the visibility setting of the called segment. For example, if the visibility attribute of a calling segment is set to OFF, all segments called from it are invisible, regardless of how they have been defined.

Changing the Attributes of a Segment

After you create a segment, you might need to alter its attributes. For example, if you created a segment using the default attributes and you want to perform a correlation operation on the subpicture in that segment, you will need to set the detectable attribute using GpiSetSegmentAttrs. You can retrieve the values of the attributes for any segment using GpiQuerySegmentAttrs.

You can change the default segment-attribute settings globally for a single presentation space using `GpiSetInitialSegmentAttrs`. The attribute setting that you specify in this function applies to all segments created subsequently in that presentation space, except that a nonretained segment can never have the dynamic attribute. For example, if you want all segments to be detectable, you can call this function to change the setting for all of them *before* you create them. `GpiSetInitialSegmentAttrs` cannot be used to change the attributes of existing segments.

You also can change the attributes of any single retained segment, but not those created subsequently, using `GpiSetSegmentAttrs`. This is useful if, for example, you want most of the segments in a picture chain to be detectable. You can change the attribute setting to detectable for the entire chain, before creating it, using `GpiSetInitialSegmentAttrs`; then, change the attribute to nondetectable for each of the segment exceptions using `GpiSetSegmentAttrs`. You can also use `GpiSetSegmentAttrs`, for example, to make a visible segment invisible when an erase request is received from the operator, or to take a segment out of the chain by redefining it as unchained. If you change an unchained segment to chained, it is added to the end of the segment chain. If you want the newly chained segment to be positioned elsewhere in the chain, use `GpiSetSegmentPriority` rather than `GpiSetSegmentAttrs`.

Some of these segment attributes apply equally to primitives that are outside of segments, although their default settings cannot be changed. Primitives outside of segments are always detectable and visible. They cannot be dynamic, because the dynamic attribute applies only to retained graphics. The chained and fast-chaining attributes do not apply to primitives outside of segments.

Editing a Segment

The operating system provides segment editing functions for writing applications that allow users to edit segments or elements in a segment. For example, after performing a correlation operation using your application, a user might need to alter the elements that intersected the pick aperture. Correlation and the pick aperture are explained in Chapter 14.

You can edit the contents of any retained segment that has a unique name. You also can edit the contents of a retained zero segment that has not yet been closed. Zero segments cannot be edited after they have been closed, because you cannot refer to a zero segment when it is no longer the current open segment.

Before you can begin editing, you must set the current drawing-mode parameter to `DM_RETAIN`. You cannot edit a segment if the current drawing-mode parameter is `DM_DRAWANDRETAIN` or `DM_DRAW`. To begin editing, call `GpiOpenSegment` and specify the name of the segment you want to edit. When you are finished editing a segment, close it using `GpiCloseSegment`.

The operating system has two edit modes: *insert* mode and *replace* mode. You can set the edit modes using `GpiSetEditMode`. You can determine which mode is currently set using `GpiQueryEditMode`.

The current edit mode applies to all segments in the presentation space until you change it. The edit mode is not an attribute of a particular segment and can be changed at any time. The default edit mode, which is set when you create a presentation space, is insert mode. When you create a graphics segment, you are actually editing it in insert mode.

If the edit mode is set to insert (SEGEM_INSERT) you can insert an element at the current location of the element pointer. The operating system shifts the element that was previously at that location into the next slot, and so on, until the last element is shifted into a new, final slot. Figure 13-3 shows a segment before and after a new element is inserted at position 0, the beginning of the segment.

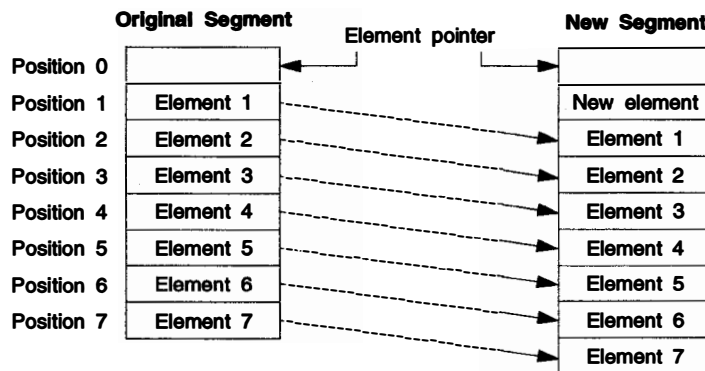


Figure 13-3. Inserting a New Element in a Segment. The new element is inserted after the current element; then, the element pointer is set to the new element.

If replace mode is set (SEGEM_REPLACE) you can replace the element at the current pointer location with a new element. Figure 13-4 shows a segment before and after the third element was replaced.

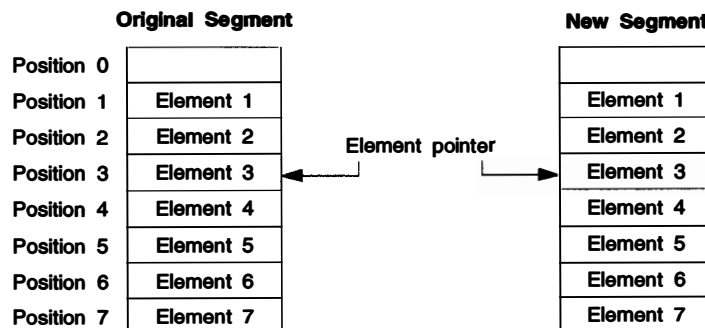


Figure 13-4. Replacing an Element with a New Element. The new element overwrites the previous element; the element pointer does not change.

Replacing elements is the recommended technique for redrawing identical primitives with different attributes, for example, color.

You can insert or replace data in an existing segment using any of the functions described in “Adding Elements to a New Segment” on page 13-4. As mentioned previously, GpiPutData is valid only in insert mode; and, only when the element pointer is addressing the final element in the segment. That is, when you are editing, you can use GpiPutData only to add data to the end of a segment.

A replace request is not valid when the element pointer is set to 0. If you call GpiOpenSegment to create a new segment without first ensuring that the current edit mode is insert, your first attempt to add an element to the segment might cause an error condition to be raised.

Deleting a Graphics Element

The PM programming interface has three different functions that allow you to delete elements within the currently opened segment.

Function	Description
GpiDeleteElement	Deletes a single element at the current element pointer location.
GpiDeleteElementRange	Deletes a range of elements in a segment.
GpiDeleteElementsBetweenLabels	Deletes a series of elements between two labels.

When you remove an element from a segment, the element pointer addresses the element immediately before the one you deleted. As with other editing functions, the current drawing mode parameter must be `DM_RETAIN` when you are deleting elements. The current editing mode (insert or replace) has no effect on the delete functions.

Deleting Graphics Segments

When you have finished drawing the subpicture associated with a segment, you should delete the segment. The PM programming interface has two different functions that allow you to delete segments.

Function	Description
GpiDeleteSegment	Deletes a retained segment
GpiDeleteSegments	Deletes a range of retained segments.

Your application can only delete segments that have unique names; because, you use the segment identifiers to identify the segment or range of segments for deletion. Retained zero segments cannot be deleted. They disappear only when their associated presentation space is deleted or reset. The presentation space can be reset either by `GpiResetPS` with the `GRES_SEGMENTS` flag or the `GRES_ALL` flag set, or by `GpiSetPS`. Resetting the presentation space is described in "Reusing the Presentation Space" on page 12-10.

If `GpiDeleteSegment` occurs within a segment bracket and the segment identified is the currently open segment, the operating system deletes the segment and ignores the remainder of the functions up to, and including, `GpiCloseSegment`. If `GpiDeleteSegment` occurs within a segment bracket and the identifier for a segment other than the currently open segment, the operating system deletes the segment, then continues processing the remaining functions in the segment bracket. If `GpiDeleteSegment` occurs outside of a segment and references a segment in the segment chain, the operating system removes the segment from the chain and links the two adjacent segments, if such segments exist.

If the range of segments deleted by `GpiDeleteSegments` were in the segment chain, the operating system "repairs" the chain by linking the segments immediately preceding and following the deleted segments, if such segments exist.

Copying a Single Graphics Element

Your application can copy the graphics orders from a single element using the `GpiQueryElement` and `GpiElement` functions. `GpiQueryElement` copies the graphics orders (or part of the orders, depending on allocated size) from an element into an array of bytes. `GpiQueryElement` is not valid within an element bracket, nor if the drawing mode is other than `DM_RETAIN`.

GpiElement copies the data from the array back into the current segment. The data may not contain any begin or end element orders. GpiElement is not valid within an element bracket. The element is retained if the drawing mode is DM_RETAIN; and drawn if the mode is either DM_DRAW, or DM_DRAWANDRETAIN.

Copying Multiple Graphics Elements

Your application can use GpiGetData and GpiPutData to:

- Copy elements from one segment to another
- Copy elements from one position in a segment to another position in the same segment.

GpiGetData copies a buffer of graphics orders from a named segment, into an area of application storage, whose byte size you specify on input. Only the named segment must not be open when you call this function. The named segment is also known as the *source segment*.

The first time GpiGetData is called, data retrieval has to start at the beginning of the segment. This is done by declaring an element offset of 0.

Output from GpiGetData depends on whether the area of application storage is large enough to hold the entire buffer of graphics orders. If so, the buffer is returned along with a count of the number of bytes of data returned to you from the segment. If not, the application storage is filled, and the count is set to the size of the application storage. The offset of the element where GpiGetData stopped copying, and the count are returned. Your application can then determine if a subsequent GpiGetData needs to be called, by checking that the count is less than the size (length) of the application storage.

On subsequent GpiGetData functions, the element offset can be 0, or it can be the offset value that was returned from the previous function. In this manner, if your application calls GpiGetData more than once to copy an entire segment, it can “pick up where it left off,” rather than recopying the segment from the beginning each time.

You can copy the data from the application storage to a new location in a segment using GpiPutData. This receiving segment is also known as the *destination segment*. The segment into which you are copying the data can be open when you call GpiPutData.

If GpiGetData does not retrieve a complete segment, the data it does retrieve can be written out to the second segment using GpiPutData, even if the last order copied is incomplete.

The current drawing mode determines whether the graphics orders are executed, stored in the segment, or both. When you call GpiPutData, the current edit mode must be SEGEM_INSERT and, if there is already data in the destination segment, the element pointer must address the last element of the segment.

Because the PM programming interface does not support explicit renaming of segments, GpiGetData—GpiPutData is the method you use to rename a segment. That is, create a second segment with the desired name, copy the contents of the first segment to it, then delete the original segment.

Drawing Retained Graphics

Nonretained graphics segments and primitives outside of graphics segment are always drawn immediately to the current output device. Segments that have been retained in segment store, however, can be drawn any number of times to any number of device contexts. The PM programming interface has three different functions that allow you to draw segments.

Function	Description
GpiDrawSegment	Draws a single segment in the chain.
GpiDrawFrom	Draws a group of segments in the chain.
GpiDrawChain	Draws the entire chain.

GpiDrawSegment draws a single, named segment, which can be chained or unchained. You supply the segment name as input to this function.

GpiDrawSegment can draw a dynamic segment in some circumstances. These are described in “Drawing Dynamic Segments” on page 13-14.

GpiDrawFrom draws one or more root segments from the segment chain. You supply the names of two root segments as input. The drawing process starts at the first named segment and draws all chained and called segments, excluding dynamic segments, up to and including the second named segment.

The order in which you specify the segment names is to reflect their relative positions in the segment chain. If the second-named segment appears in the segment chain before the first-named segment, no error is raised, but drawing starts from the first named segment and continues to the end of the chain. This also happens if the second segment is an unchained segment rather than a root segment. If the drawing control, **DCTL_DYNAMIC** is set, any dynamic segments already drawn on the display are removed before **GpiDrawFrom** begins to draw nondynamic segments; then, they are redrawn after **GpiDrawFrom** ends.

GpiDrawChain draws all nondynamic chained segments in a named presentation space, as well as any unchained segments called by those chained segments. It does not draw dynamic segments. You supply only the presentation space handle as input to this function. If the drawing control, **DCTL_DYNAMIC** is set, any dynamic segments already drawn on the display are removed before **GpiDrawFrom** begins to draw nondynamic segments; then, they are redrawn after **GpiDrawFrom** completes.

You can call any of these functions while a segment is open. The open segment is not modified by these functions, although any of them can result in the current attribute values and current position being modified. For more information about the attribute-resetting process, see Appendix C, “Graphics Attributes.”

The current drawing mode determines whether the segments identified in the drawing functions are executed, stored in the segment, or both. The current drawing mode does not effect which segments are drawn, or the priority in which they are drawn for any of these three GPI drawing functions.

If an error occurs during the drawing process, you can determine where the error occurred using **GpiErrorSegmentData**. This function returns information about the last error that occurred during a segment-drawing operation. It returns a pointer to the segment identifier and a pointer to one of three constants that indicate when the error occurred.

Constant	Error occurred...
GPIE_SEGMENT	While drawing the segment.
GPIE_ELEMENT	While calling GpiElement.
GPIE_DATA	While calling GpiPutData.

Drawing Dynamic Segments

A *dynamic segment* is a chained segment that possesses special properties. Dynamic segments are those that can be moved or changed in some other way without disturbing the remaining parts of the picture. To draw dynamic segments on the display screen for the first time, use GpiDrawDynamics. GpiDrawDynamics draws every dynamic segment in the segment chain.

When they have been drawn on the display, dynamic segments can be updated. For example, to move one or more dynamic segments to another part of the display screen, remove them from their current position using GpiRemoveDynamics. You can restrict the scope of this function by supplying the names of a start segment and a finish segment. The start segment and finish segment can be the same. Dynamic segments outside of this range are not removed from the display screen.

If you intend to re-associate the presentation space, call GpiRemoveDynamics first; otherwise, GpiRemoveDynamics cannot remove the dynamic segments after the presentation space has been re-associated. The dynamic segments effectively cease to be dynamic.

After removing the appropriate dynamic segments from the display screen, you can redraw them elsewhere, again using GpiDrawDynamics. If you specify a range of dynamic segments in GpiRemoveDynamics, the redrawing is restricted to that range. Note that, when dynamic segments have been drawn on the display screen, you must not edit their definitions in segment store.

Dynamic segments are treated like nondynamic segments when you are directing output to a hardcopy device. GpiRemoveDynamics and GpiDrawDynamics raise an error condition if the current output device is not a display window.

The Drawing Controls

GpiErase clears the output display of the currently associated device to CLR_BACKGROUND, which is the normal background color for the device. If you are using the default color table, this value clears the window background to white. This is a once-only request and must be issued each time you want the display screen cleared before drawing a new picture. You can use this function for either a micro presentation space or a normal presentation space.

To get this effect for more than one drawing request, you can use GpiSetDrawControl. This function establishes current values for five drawing controls, which remain in effect until they are reset. Table 13-4 on page 13-15 describes the five drawing controls.

Table 13-4. Drawing Controls

Control	Value	If set, the operating system...
Boundary data accumulation	DCTL_BOUNDARY	computes the dimensions of the smallest rectangle that would completely surround the retained-drawing output. This control is described in Chapter 16, "Clipping and Boundary Determination."
Correlation	DCTL_CORRELATE	performs correlation operations on any primitives or any output associated with GpiPutData or GpiElement. This control is described in Chapter 14, "Correlation."
Display control	DCTL_DISPLAY	draws retained output on the device identified by the current device context. If this control is not set, no retained output will appear on the device.
Draw-dynamic-segments	DCTL_DYNAMIC	calls GpiRemoveDynamics before drawing any retained output and then, after drawing the retained output, it calls GpiDrawDynamics to draw output stored in dynamic segments. This automatically removes all dynamic segments from the display screen before a GpiDrawxxxxx request is issued, then later redraws the segments. This ensures that dynamic segments are always drawn on top of nondynamic segments and primitives.
Erase-before-draw	DCTL_ERASE	calls GpiErase before drawing any retained output.

GpiSetDrawControl can be called in either a micro or a normal presentation space, although not all of the controls are valid in a micro presentation space. For details, see the *Presentation Manager Programming Reference*. GpiDrawSegment can be used to draw a dynamic segment in some circumstances. Its effects are as follows:

- If the named segment is both chained and dynamic, and the DCTL_DYNAMIC drawing control is set, *all* dynamic segments in the chain, including the named segment, are drawn as dynamic.
- If the named segment is both chained and dynamic, and the DCTL_DYNAMIC control is not set, the named segment is not drawn.
- If the named segment is unchained and dynamic, it is drawn as a nondynamic segment, regardless of the setting of the DCTL_DYNAMIC control.

If you called GpiRemoveDynamics prior to calling GpiDrawDynamics and you specified a range of dynamic segments, the operating system draws only that range. If you set the DCTL_DYNAMIC control using GpiSetDrawControl, the operating

system calls `GpiRemoveDynamics` before drawing the subpictures from the dynamic segments.

The `DCTL_DISPLAY` control is the only control set to `DCTL_ON` by default. All other controls are set to `DCTL_OFF` when you create a presentation space.

Using Segment Editing Functions

You can use editing functions to:

- Create a chained-dynamic segment
- Delete a segment
- Edit the contents of a segment.

Figure 13-5 on page 13-17 shows an example of how to edit the contents of a segment. In the example, three elements are inserted. The steps required for this task are as follows:

1. Set the segment edit mode to insert or replace using `GpiSetEditMode`.
2. Set the drawing mode to retain.
3. Open the segment using `GpiOpenSegment`, passing it the segment identifier from a previous correlation operation.
4. Set the element pointer so that it points to the position at which you will replace or insert an element using `GpiSetElementPointer`, `GpiSetElementPointerAtLabel`, or `GpiOffsetElementPointer`.
5. Insert the new primitives using any of the Gpi primitive functions.
6. Delete any unnecessary primitives using `GpiDeleteElement` or `GpiDeleteElementRange`.
7. Close the segment using `GpiCloseSegment`.

The first element inserted contains the graphics order that sets the color to yellow; the second element moves the current position; and the third element draws an outlined box with rounded corners. After the three elements are inserted, the code deletes the element at position 5 in the segment (this element was previously at positions 1 and 2).

```

#define INCL_GPISEGEDITING
#define INCL_GPICONTROL
#include <os2.h>
void fncESEG01(void){
    HPS hps;
    LONG idNonChained;
    POINTL ptl;

    GpiSetEditMode(hps, SEGEM_INSERT);

    GpiSetDrawingMode(hps, DM_RETAIN);

    GpiOpenSegment(hps, idNonChained);

    GpiSetElementPointer(hps, 0L);

    GpiSetColor(hps, CLR_YELLOW);

    ptl.x = 100; ptl.y = 100;

    GpiMove(hps, &ptl);

    ptl.x = 150; ptl.y = 150;

    GpiBox(hps, DRO_OUTLINE, &ptl, 40L, 40L);

    ptl.x = 30; ptl.y = 30;

    GpiMove(hps, &ptl);

    GpiSetElementPointer(hps, 5L);

    GpiDeleteElement(hps);

    GpiCloseSegment(hps);
} /* fncESEG01 */

```

Figure 13-5. Editing the Contents of a Graphics Segment

Summary

Table 13-5 summarizes the functions used to edit retained graphics and segments.

<i>Table 13-5 (Page 1 of 2). Editing Graphics and Segment Functions</i>	
Function Name	Description
GpiBeginElement	Defines the beginning of an element bracket.
GpiCloseSegment	Defines the end of a segment bracket.
GpiDeleteElement	Deletes the element at the element pointer, then sets the element pointer to the previous element.
GpiDeleteElementRange	Deletes a specified range of elements from a segment.
GpiDeleteElementsBetweenLabels	Deletes all of the elements in a segment that appear between two labels.
GpiDeleteSegment	Deletes a segment.
GpiDeleteSegments	Deletes a specified range of segments.
GpiDrawChain	Draws the subpictures stored in a presentation space's segment chain.
GpiDrawDynamics	Redraws dynamic segments.
GpiDrawFrom	Draws the subpictures from a specified range of segments in the segment chain.
GpiDrawSegment	Draws a subpicture from a single segment.
GpiErase	Clears a window identified by a screen device context and paints the window, using the color identified by index 0 in your presentation space's color table.
GpiElement	Creates an element from graphics orders that you store in a buffer and pass to the function.
GpiEndElement	Defines the end of an element bracket.
GpiErrorSegmentData	Provides information about the last error that occurred during a retained-drawing operation.
GpiGetData	Copies graphics orders from a segment into a buffer.
GpiLabel	Generates a special element called a label.
GpiOffsetElementPointer	Moves the element pointer in a segment by a specified offset.
GpiOpenSegment	Defines the beginning of a segment bracket.
GpiPutData	Copies graphics orders from a buffer into a segment.
GpiQueryDrawControl	Determines whether one of the five drawing controls is set.
GpiQueryDrawingMode	Determines which of the three drawing modes is set.
GpiQueryEditMode	Determines which of the two segment-editing modes is currently set.
GpiQueryElement	Retrieves the graphics orders from the element at the current position of the element pointer and copies the orders into a buffer of bytes.

Table 13-5 (Page 2 of 2). Editing Graphics and Segment Functions

Function Name	Description
GpiQueryElementPointer	Retrieves the location of the element pointer.
GpiQueryElementType	Retrieves the type of the element at the current location of the element pointer.
GpiQueryInitialSegmentAttrs	Retrieves the setting of one of the seven initial segment attributes.
GpiQuerySegmentAttrs	Retrieves the setting of one of the seven segment attributes.
GpiQuerySegmentNames	Retrieves a list of existing segment identifiers within a specified range.
GpiQuerySegmentPriority	Returns the identifier for a segment that precedes or follows a segment in the segment chain.
GpiQueryStopDraw	Determines whether the stop-draw condition is set.
GpiRemoveDynamics	Removes parts of a picture that were drawn by dynamic segments in the segment chain.
GpiSetDrawControl	Sets one of the five drawing controls.
GpiSetDrawingMode	Sets the drawing mode in your presentation space to one of three possible modes: draw, retain, or draw-and-retain.
GpiSetEditMode	Defines which of the two segment-editing modes is currently set.
GpiSetElementPointer	Sets the element pointer so that it points at the nth element in a segment.
GpiSetElementPointerAtLabel	Sets the element pointer to the element identified by a particular label.
GpiSetInitialSegmentAttrs	Sets the default segment attributes.
GpiSetSegmentAttrs	Sets one of the seven segment attributes.
GpiSetSegmentPriority	Alters the order in which the operating system draws and detects segments in the segment chain.
GpiSetStopDraw	Sets the stop-draw condition.

Chapter 14. Correlation

Correlation is the process of determining which primitives, if any, are contained in an area of interest. The process for nonretained graphics is very different from the process for retained graphics. The following topics are related to the information in this chapter:

- Presentation spaces
- Coordinate spaces and transformations
- Segments.

About Correlation

When you want to select an area of interest on the screen, you usually move the pointer to the applicable point and signal (by clicking a mouse, for example) that this is the object you want. This selection process most commonly is used for graphic applications. For example, your application could permit a user to select an object, then change its color. Correlation, however, also can be used in nongraphic applications. For example, your application models the operation of a calculator and permits a user to select numbers for mathematical operations.

The *area of interest* is defined by the operating system as a small rectangle, centered on the (x,y) coordinate position, that has been sent to the application. The virtual rectangle the application generates is known as the *pick aperture*.

Figure 14-1 shows an example of a pick aperture.

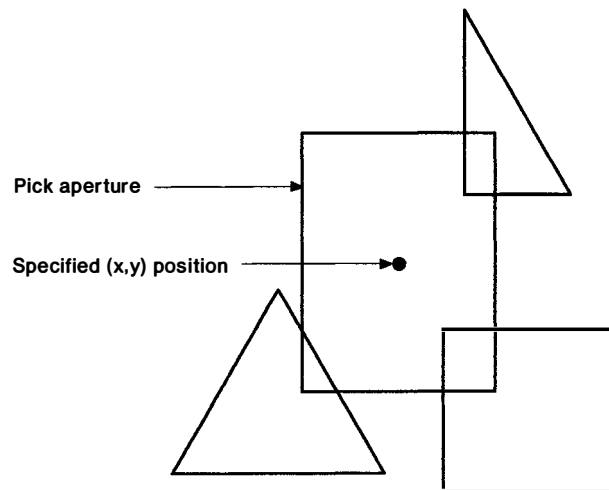


Figure 14-1. The Pick Aperture

The process of determining what lies in the pick aperture differs between nonretained and retained graphics. For nonretained graphics, the presentation space must be put in a state that supports correlation before the primitives are drawn; then, correlation is performed while the primitives are being drawn. For retained graphics, the segments that contain the graphics can be replayed, permitting correlation after the primitives are drawn.

Note: The necessity to correlate graphics usually can be predicted; therefore, you can plan for it when designing applications. However, this does not mean that you have to create retained (rather than nonretained) graphic segments. The decision to retain graphics is dependent on several considerations of which correlation is only one.

Correlating Nonretained Graphics

For the purposes of correlation, *nonretained graphics* are those graphics that are being correlated *during* the drawing process. Nonretained graphics can exist in nonretained graphic segments or completely outside any segment structure. Primitives outside segments are detectable when the applicable draw control is set.

Nonretained graphics, inside a segment bracket, can be created in either *draw* or *draw-and-retain* modes. If created in draw-and-retain mode, a segment, at first, is considered nonretained while the primitives in the segment are being drawn; then, it is considered retained. To be correlated, nonretained segments must have unique, nonzero identifiers, and must be defined as *detectable*. The primitives within these segments can be *tagged* just as primitives in retained segments are; however, the tags do not influence the correlation process for nonretained graphics.

To get correlation data from the drawing of nonretained graphics, 3 steps must be performed— after creation of the presentation space but before drawing the primitives:

1. Call `GpiSetDrawControl` to switch on the *correlation flag* (`DCTL_CORRELATE`, `DCTL_ON`).
2. Call `GpiSetPickApertureSize`, if necessary, to change the size of the pick aperture.
3. Call `GpiSetPickAperturePosition`, if necessary, to explicitly position the aperture. As input to this function, you provide the coordinate position on which the pick aperture is to be centered, using presentation page coordinates.

Correlation is performed for the following functions:

- Individual primitive-drawing requests, for example, `GpiBox`
- `GpiPutData`
- `GpiElement`
- `GpiPlayMetaFile`.

Correlation is never performed for `GpiErase`.

You detect correlation *hits* by examining the returned values from the GPI functions. If `GpiLine`, for example, draws a line that intersects the pick aperture, it returns a value of `GPI_HITS` to indicate a correlation hit. If the line does not intersect the pick aperture, `GpiLine` returns a value of `GPI_OK`, to indicate the successful drawing of a line without a correlation hit. `GpiLine` returns a value of `GPI_ERROR` if an error is detected.

If the line intersects the pick aperture, a correlation hit is returned even if the line style is `LINETYPE_INVISIBLE`. For other primitives, if the object is drawn in outline mode, a correlation hit is returned only if the pick aperture intersects the boundary. If the object is in fill mode, a correlation hit is returned if the pick aperture intersects or lies within the boundary.

Figure 14-2 is an example of primitives intersecting the pick aperture.

Correlation on Non-retained Images

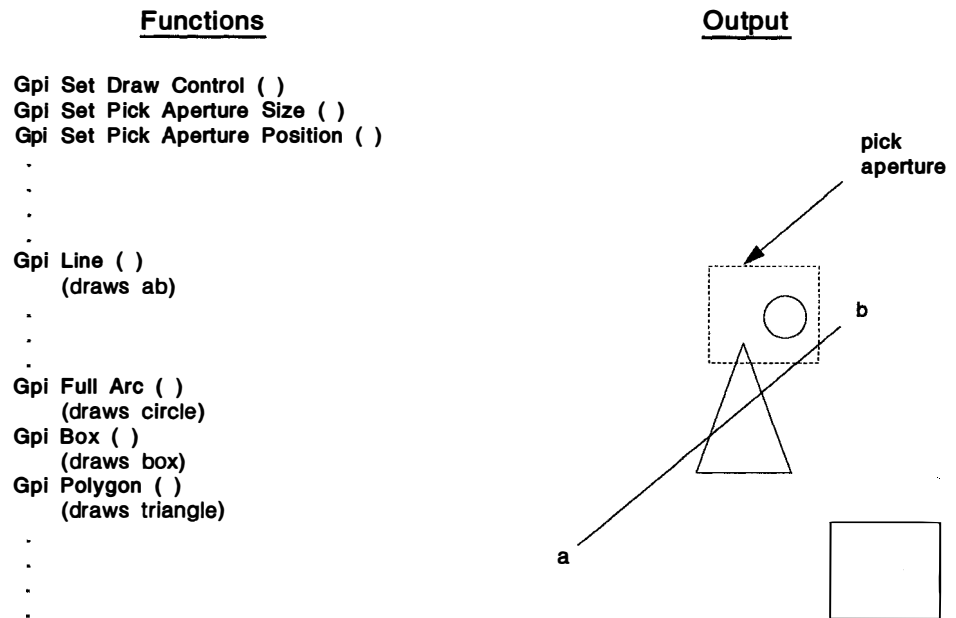


Figure 14-2. Correlating Nonretained Graphics. Each GPI function whose output intersects the pick aperture returns a hit (GPI_HIT). GpiBox, whose output does not intersect the pick aperture, returns GPI_OK.

Correlating Retained Graphics

The information in a retained graphic segment can be redrawn; therefore, setting up an environment for correlation is not necessary. Correlation on retained segments is processed through segment IDs and tags throughout the segments.

For any retained segment to be a candidate for correlation, the following must be done:

1. Call GpiOpenSegment with a unique, nonzero identifier.
2. Call GpiSetSegmentAttrs with the following settings:
 - a. ATTR_DETECTABLE switched ON
 - b. ATTR_DYNAMIC left OFF (default),
unless the segment is unchained and, therefore, drawn non-dynamically.
 - c. ATTR_PROP_DETECTABLE left ON (default),
in any segment that calls the candidate segment.
3. Call GpiSetTag at appropriate locations within the segment

Note: The preceding is the recommended method. However, an application still can receive correlation data about invisible segments, as explained in “The IType Input Parameter” on page 14-5

Tagging Primitives within a Segment

The `GpiSetTag` function inserts a long integer value, called a *tag*, at the current element pointer position. The tag becomes a segment element, which serves as an identifier for the primitives that follow until the next tag is issued. Tags also are called *pick tags* or *pick identifiers*. Typically, applications assign tags only to elements that correspond to primitives. You can determine the value of the last tag assigned to an element using `GpiQueryTag`.

Note: Tags are used only in correlation; they are not used in the chaining or calling of segments, nor do they cause or modify output.

The long integer value of the tag is 0 by default. However, a 0 value makes the subsequent primitives undetectable. An application can use this effect to make some parts of a segment detectable and other parts of it non-detectable. If you know that this capability is unnecessary, you can change the default tag value using `GpiSetDefTag`.

The tag you specify becomes the current tag, and it applies to all subsequent primitives until you next call `GpiSetTag`. The tag can be considered one of the attributes of the primitive, and, therefore, affected by the current attribute mode. In `AM_PRESERVE` mode, it is stored on the LIFO stack and can be recalled with `GpiPop`. Tags cannot be inserted between a `GpiBeginArea` and `GpiEndArea` area bracket; therefore, all primitives within an area have the same tag.

A tag value greater than 0 enables correlation on the subsequent primitives, but only if the segment ID is greater than 0 as well. The data returned from each correlation consists of a set of segment-tag pairs. The reason for this pairing is that a single segment can draw objects in several locations. By adding identifiers within the segment, an application has a more exact description of what has intersected the pick aperture.

For simple chained segments, each unique segment-tag pair within the pick aperture is known as a *hit*. If two or more primitives in the pick aperture have the same tag, they are considered a single hit. Hits for called segments differ slightly and are described in "The `IMaxDepth` Input Parameter" on page 14-8.

Correlation Functions for Retained Graphics

The PM programming interface has 3 different functions that enable you to correlate a uniquely identified retained segment.

Function	Description
<code>GpiCorrelateSegment</code>	Permits correlation on a single segment.
<code>GpiCorrelateFrom</code>	Permits correlation on a range of segments from a segment chain.
<code>GpiCorrelateChain</code>	Permits correlation on the entire segment chain.

For nonretained graphics, the correlation hits are returned by the actual drawing commands. For retained graphics, the correlation hits are returned by the 3 correlation functions, listed above.

The size of the pick aperture is set using `GpiSetPickApertureSize`, just as with nonretained graphics. However, the coordinate position on which the pick aperture is centered usually is obtained from operator input, for example, from a `WM_BUTTON1DOWN` message, instead of from `GpiSetPickAperturePosition`.

After the graphics orders that create pictures are stored in retained segments, the pictures can be re-created by your application with the various GpiDrawxxx functions. Then the user can view the pictures, if the output is directed to a screen for example. After the user selects an area of interest, a GpiCorrelatexxx function redraws the picture internally to determine just what intersects the pick aperture. The user does not see the re-creation. A standard order of functions within your application would be:

- GpiDrawxxx
- GpiSetPickApertureSize
- GpiCorrelatexxx.

The Correlation Input Parameters

There is only one segment chain per presentation space; therefore, GpiCorrelateChain needs the presentation space handle as input. The segment correlation functions need both the presentation space handle and the segment IDs.

All three correlation functions require the following as input:

- Correlation attribute type
- Maximum number of hits
- Number of segment-tag pairs to be returned for a single hit, called the *pair depth*.

The IType Input Parameter: The two classifications of segments upon which you can request correlation are the following:

- Segments that have been defined as both detectable and visible
- All nonzero segments, regardless of their detectability and visibility attributes.

The IMaxHits Input Parameter: The correlation functions return the number of hits made on the retained segments. Figure 14-3 on page 14-6 is an example of a line intersecting the pick aperture.

Correlation on Retained Segments

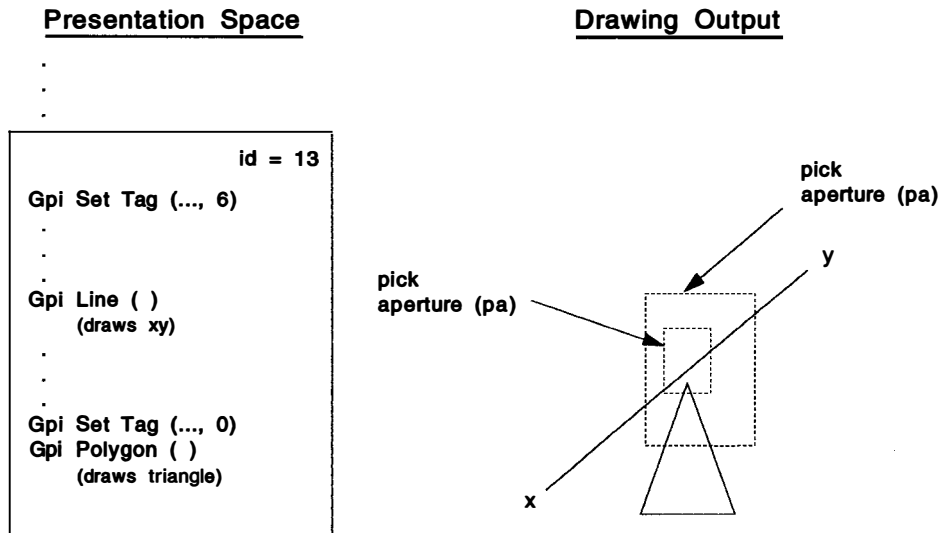
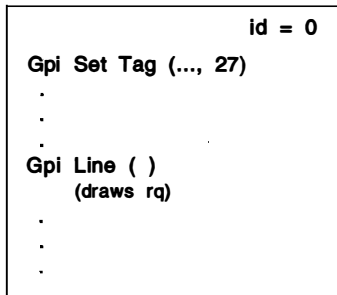
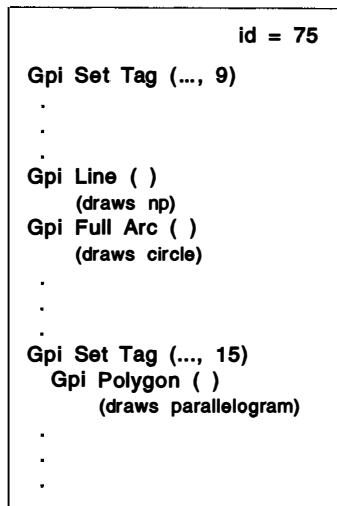


Figure 14-3. Retained Segment Correlation with One Hit. The intersection of a unique segment-identified and -tagged primitive with the pick aperture (pa) produces one hit. If the pick aperture size were increased, there still would be only 1 hit. The triangle produces no hit because its tag is 0.

Figure 14-4 on page 14-7 is an example of multiple primitives intersecting the pick aperture.

Correlation on Retained Segments

Presentation Space



Drawing Output

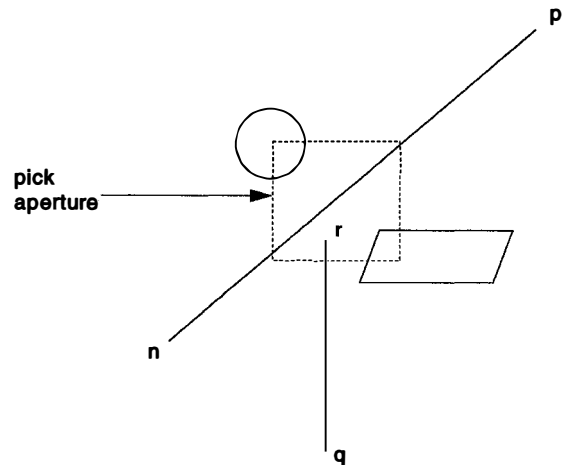


Figure 14-4. Retained Segment Correlation with Multiple Hits. Four separate primitives intersect the pick aperture; however, since 2 primitives share the same tag, and 1 primitive has a segment ID of 0, there are only 2 hits.

Your application can set a limit on the number of hits to return from a correlation function. The maximum-number-of-hits parameter influences the size of the array created to handle the segment-tag pair returned for each hit. By comparing the maximum number desired to the actual number of hits, your application can determine whether all hits are accounted for. Figure 14-5 on page 14-8 shows an example of the `alSegTag` data structure that contains the segment-tag pairs for Figure 14-4.

Correlation on Retained Segments

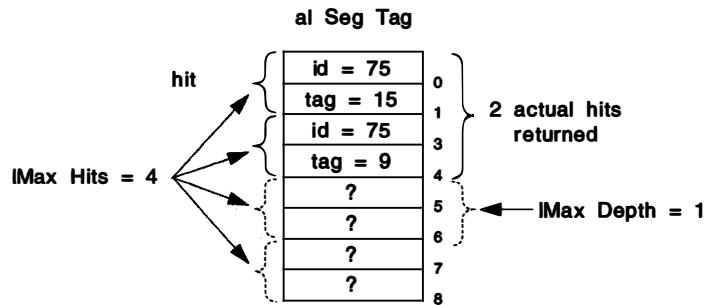


Figure 14-5. `alSegTag` Data Structure

As shown in Figure 14-5, the identifier—tag pairs are returned to the application in the reverse order of their occurrence on the segment chain. That is, the highest priority segment is returned first. Therefore, the application can identify the topmost segment, which is the segment most likely to have been picked.

The `IMaxDepth` Input Parameter: When a called segment is picked, correlation data is returned also for all segments above it in the hierarchy, up to and including the root segment. Figure 14-6 on page 14-9 is an example of a picture drawn from a complex segment chain with called segments.

Correlation on Retained Segments

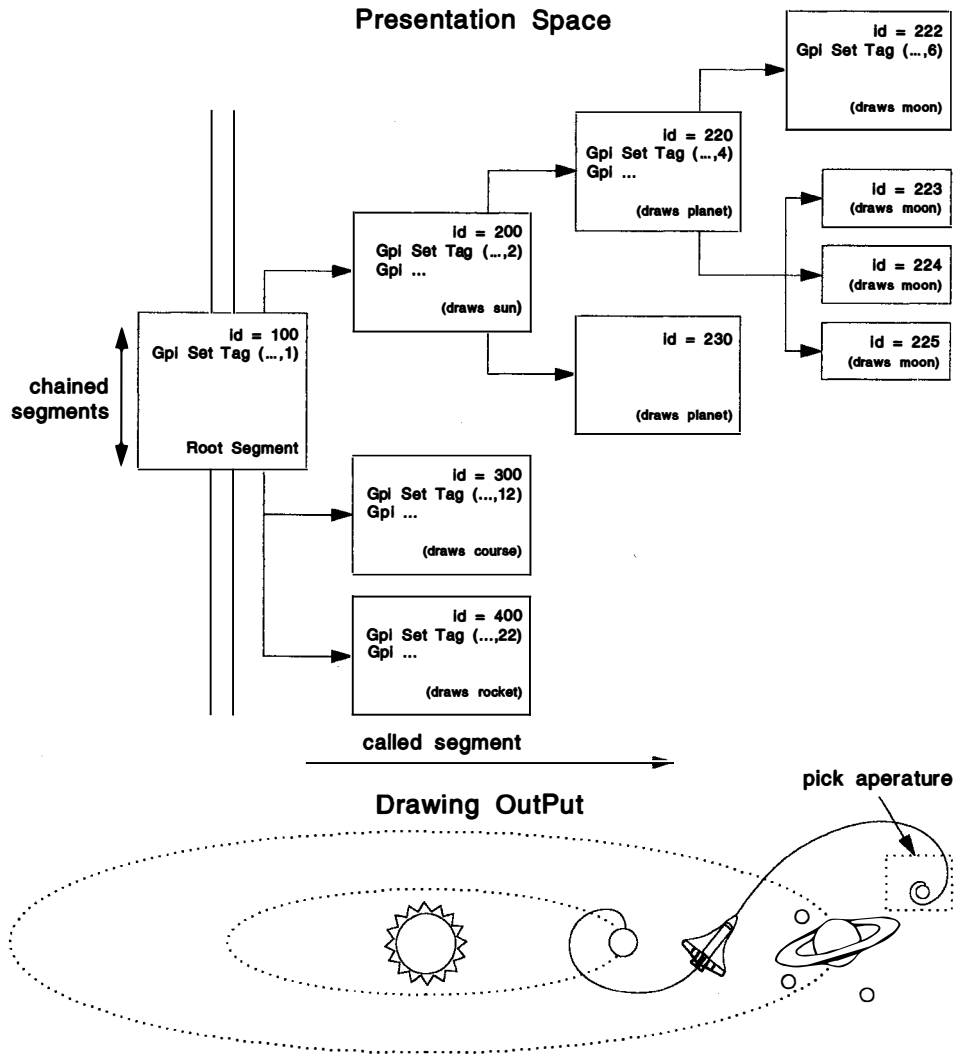


Figure 14-6. Multiple Hits from a Called Segment. Two separate items, called from different portions of the segment chain, intersect the pick aperture. Each has a unique segment identifier and tag, so there are two hits.

For called segments, the group of segment-tag pairs constitutes a single hit. You can limit the number of segment and tag pairs returned for each hit using the maximum depth parameter, just as you can limit the total number of hits returned to you using the maximum number of hits parameter. Figure 14-7 on page 14-10 shows two examples of the `alSegTag` data structure from Figure 14-6, for two different `IMaxDepth` values.

Correlation on Retained Segments

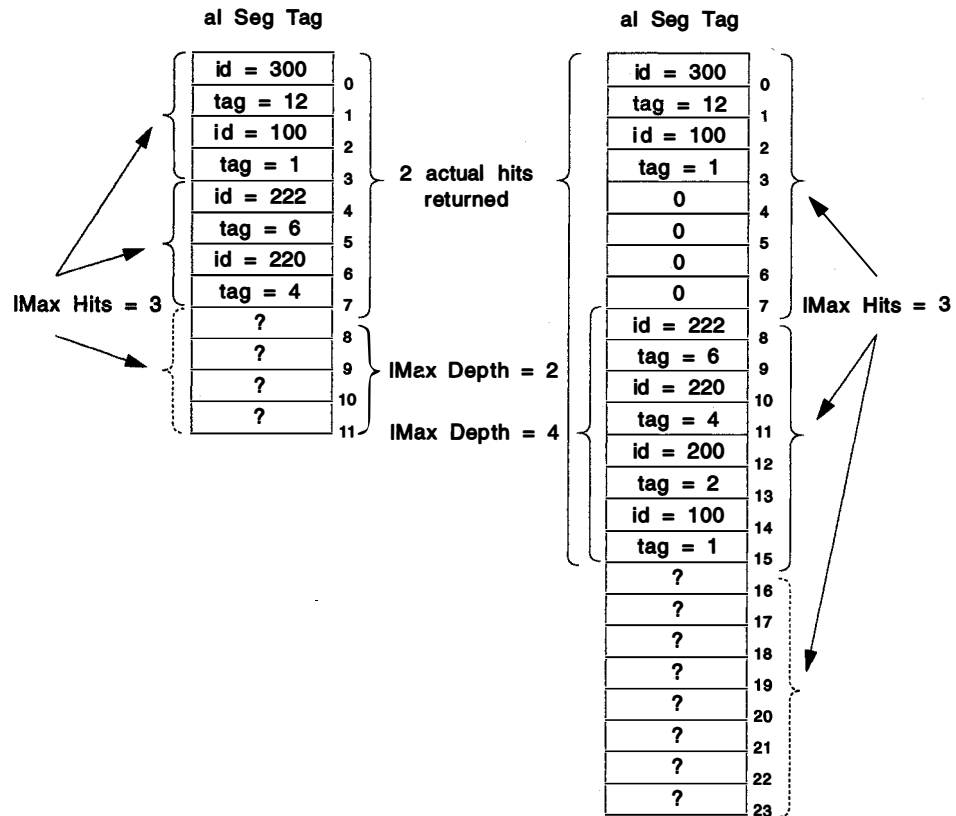


Figure 14-7. `alSegTag` Data Structure for Different `IMaxDepth` Values. Unused segment-tag pairs for actual hits are set to 0 in the `alSegTag` array.

There are two major reasons for the PM programming interface to provide this capacity. The first is the consideration of application storage. If your application is a graphics package, for example, providing extensive design capabilities to an end user, the user's drawing may be very complex with 10 or more levels of segment calling. The data returned from a single hit could require an `alSegTag` array so large the data overruns the application storage you had reserved. By setting a maximum calling depth, your application can reserve the correct amount of storage.

The second consideration is the knowledge that your application's user is interested only in a certain level of calling depth. Many users will be interested only in the topmost called segment, because it usually is the segment containing the functions that performs the actual drawing.

Pick Aperture

Your application determines the size of the pick aperture using `GpiQueryPickApertureSize`, and determines the page space coordinates of the center of the aperture with `GpiQueryPickAperturePosition`.

Because your objective is to retrieve a single detectable object for each correlation operation, you have to obtain a balance between the number of detectable objects in a picture and the size of the pick aperture. The more detectable objects you define, the smaller the pick aperture must be to return a manageable amount of information to the application. However, the pick aperture usually is not a 1 pel-by-1 pel rectangle. A larger rectangle makes better sense because the pick aperture can be difficult to hit with a single pel; but a larger sampling area, for example 4 pels-by-4 pels, increases the probability of an intersection between a lighted pel and the pick aperture.

The pick aperture can be specified in any of the units available to the presentation space. However, the type of unit must be identical to the type specified for the presentation page. The default size of the pick aperture is square, with sides equal to the default character cell height.

Using Correlation

This section explains how to perform correlation on the graphics associated with a segment.

Figure 14-8 shows how to set the pick aperture size with `GpiSetPickAperture`, and correlate the segment chain with `GpiCorrelateChain`, passing it the pick-aperture position as the third argument. The functions use the `psizlPick` and `pptlPick` arguments to set the aperture size and position the aperture center. The correlation is performed on visible and detectable segments only (`PICKSEL_VISIBLE`). `GpiCorrelateChain` copies any segment-tag pairs to the buffer pointed to by `a1SegTag` and returns the count of hits detected.

```
#define INCL_GPICORRELATION
#include <os2.h>
LONG fncCORL01 (HPS hps, PSIZEL psizlPick, PPOINTL pptlPick,
               LONG lMaxHits, LONG lMaxDepth, PLONG a1SegTag)
{
    LONG cHits;

    GpiSetPickApertureSize(hps, PICKAP_REC, psizlPick);
                                /* Set the pick aperture. */

    cHits = GpiCorrelateChain(hps, PICKSEL_VISIBLE, pptlPick,
                             lMaxHits, lMaxDepth, a1SegTag);
                                /* Correlate the hits. */

    return (cHits);
                                /* Return count of hits. */
} /* fncCORL01 */
```

Figure 14-8. Performing a Correlation Operation

Summary

Table 14-1 summarizes the functions used to correlate both retained and nonretained graphics and segments.

<i>Table 14-1. Correlation Functions</i>	
Function Name	Description
GpiCorrelateChain	Determines whether correlation hits occurred at the current pick-aperture location for any segment either actually in the segment chain or called by a segment in the chain.
GpiCorrelateFrom	Determines whether correlation hits occurred at the current pick-aperture location for any segment in a range of chained segments.
GpiCorrelateSegment	Determines whether correlation hits occurred at the current pick-aperture location for a given segment.
GpiErase	Clears a window identified by a screen device context, and paints the window, using the color identified by index 0 in the presentation space color table.
GpiQueryBoundaryData	Retrieves the size of the smallest rectangle that completely surrounds the current segment output, if the drawing control DCTL_BOUNDARY has been set by calling GpiSetDrawControl.
GpiQueryPickAperturePosition	Retrieves the location of the center of the pick aperture in the application's page space.
GpiQueryPickApertureSize	Retrieves the dimensions of the pick aperture in presentation page coordinates.
GpiQueryTag	Retrieves the value of the last tag set by calling GpiSetTag.
GpiResetBoundaryData	Resets the boundary data to NULL.
GpiSetDrawControl	Sets one of the five drawing controls. These controls are described in the Table 13-4 on page 13-15.
GpiSetPickAperturePosition	Sets the location of the center of the pick aperture in the application's page space.
GpiSetPickApertureSize	Sets the dimensions of the pick aperture in presentation page coordinates.
GpiSetTag	Assigns a tag to an element in a segment.

Chapter 15. Metafiles

A metafile is a graphics object that, like a segment, contains the GPI instructions that contribute to the final version of a picture. Unlike a segment, a metafile also contains a header, with state information, and all resources necessary to identically create the picture. Because pictures that are displayed when a graphics application is executed are lost when the application finishes, metafiles provide a method of retaining pictures beyond a single execution of an application.

The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Line primitives
- Marker primitives
- Area primitives
- Character string primitives
- Color and mix attributes
- Paths
- Regions
- Fonts
- Printing
- Segments
- Bit maps
- Coordinate spaces and transformations.

About Metafiles

Metafiles can exist in three distinct forms. A metafile that has just been created is called a *memory metafile* because it exists in memory managed by the Presentation Manager on behalf of the application that created it. A metafile that is transferred to disk storage as a file with the default extension of .MET is called a *disk metafile*. A metafile that is loaded into an application's memory is *editable* by the application.

Metafiles save resources in the following ways:

- Metafiles can be created and used in draw mode by a single application. Therefore, an application that is not retaining segments in a segment store can retain graphics in a metafile.
- Metafiles remain available while the owning application is running, regardless of the number of presentation spaces the application obtains or defines.
- Given the same starting conditions in each presentation space, you can produce an identical picture each time the metafile contents are executed.
- Different threads or processes within an application can display a picture stored as a metafile without each having to own the metafile.
- If your organization has common graphical resources, such as a company logo, those resources can be stored in a metafile. This avoids the overhead of re-creating the resources each time they are needed.

- Because the metafile is a device-independent format, it is useful for transferring pictures that are to be printed when the printer-type is unknown.
- PM applications can exchange graphical information by using metafiles, either by using the clipboard or by transferring them over a network.
- Your application can also exchange graphics data with non-PM applications that support the Mixed Object Document Content Architecture (MO:DCA) interchange standard.

Unlike bit-maps, metafiles offer some device-independence. Bit maps store picture information on a pel-by-pel basis. Metafiles store picture information in the form of low-level graphics commands that the operating system uses to construct the pictures.

Note: Metafiles can contain bit maps or other graphics information that is in device-dependent format.

The graphics commands, called “graphics orders,” represent graphics functions that create a picture. These include drawing instructions, as well as attribute-setting instructions (for example, color tables and logical fonts) and anything that describes the structure of the picture. The contents of a metafile, therefore, are similar to those of the graphics presentation space in which the picture is drawn. The Presentation Manager automatically records the environmental detail of the presentation space in which a picture is drawn in the metafile.

Note: A metafile can contain data generated from GPI functions only. Any non-graphical data included in a metafile is ignored.

An application can re-create a picture from a metafile and display it in a window on the screen or print it by “playing” the metafile. When an application displays the contents of a metafile, it can use the color table, font, fill pattern, and transformations that are stored in the metafile, or it can use the logical color table, logical font, fill pattern, and transformations that are set for the current presentation space. The appearance of the picture stored in the metafile can, therefore, be changed by editing the current presentation space before playing the metafile.

An application can save metafiles to a disk to be loaded later by any application that chooses to access the metafile. Disk metafiles loaded into application memory can also be edited in the same manner as elements in graphics segments are edited.

Contents of a Metafile

Every graphic function can be represented by one or more graphics orders. The operating system defines graphics orders for:

- Areas
- Bit maps (for fill patterns) and pattern-reference points
- Character output and attribute
- Colors and mix modes (foreground and background)
- Lines and arcs and line and arc attributes (including GpiBox)
- Paths (including clip paths)
- Position (for example, GpiMove)
- Saving and resetting attributes (including GpiPop)
- Transformations (in model space)
- Viewing Limits.

Chapter 13, "Editing Retained Graphics and Graphics Segments" also describes orders and how to edit them.

When a picture is drawn in a presentation space associated with a metafile device context, the operating system converts the primitive attributes and drawing functions into graphics orders and stores them in the metafile.

The operating system stores the effects of region and other operations that are not permitted as escape orders in the metafile. When an application finishes playing a metafile, the system restores any regions that were defined for the presentation space prior to calling `GpiPlayMetaFile`.

An application can use any of the query functions while creating a metafile, but the system will not store those query functions in the metafile. Similarly, an application can set a value by using a mathematical operator, but the system will store only the result of the operation in the metafile. For example, if an application determines the end point of a line by subtracting 100 from the x- and y-coordinates of the upper-right corner of a window, the system records the end point, not the relative distance, in the metafile.

Because the output of bit map functions is dependent upon the capabilities of the output device, Gpi bit map functions might produce unexpected results when they are used in a metafile. If the pel dimensions of the output device are different from the pel dimensions of the device on which the metafile was created, the bit map will be distorted. If the output device is a plotter, the bit map functions will not work at all.

Most of the escape functions used by the `DevEscape` function can be used in metafiles. The escape functions that the system does not save in metafiles are `DEVESC_QUERYESCSUPPORT` and `DEVESC_GETSCALINGFACTOR`.

Metafile Content Restrictions

Like areas and paths, not all functions and attributes can be represented in a metafile. Metafile content is mainly dependent upon the drawing mode in which the metafile is to be played. If the metafile is to be played when the current drawing-mode parameter is `DM_DRAW` (and in no other drawing mode), the metafile content is restricted as shown in "Draw-Mode Restrictions." If the metafile is to be played when the current drawing-mode parameter is `DM_RETAIN` or `DM_DRAWANDRETAIN`, or if the file is to conform to SAA* guidelines, the metafile content is restricted as shown in "Creating Metafiles for Interchange."

Metafile restrictions are also described in Appendix G in *Presentation Manager Programming Reference Volume III*.

Draw-Mode Restrictions: The following restrictions apply if the metafile is to be played when the current drawing mode parameter is DM_DRAW:

- If the DCTL_DISPLAY flag of GpiSetDrawControl is OFF when you are sending graphics to a metafile, the graphics will not be visible when the metafile is played. Their effects on the current position and on current attribute settings are recorded.
- Region functions are not recorded in a metafile. The effects of the following functions, however, are recorded:
 - GpiSetClipRegion
 - GpiIntersectClipRegion
 - GpiExcludeClipRegion
 - GpiOffsetClipRegion
 - GpiPaintRegion.

When the metafile is played into a presentation space, temporary regions are created. Upon completion of GpiPlayMetaFile, these temporary regions are automatically deleted and the clipping region that was current before the metafile was played, is restored.

- In general, escape functions identified by reserved escape codes (that is, escape codes in the range of 0 through 32767) are recorded in a metafile. However, the DevEscape functions DEVESC_QUERYESCSUPPORT (escape code 0) and DEVESC_GETSCALINGFACTOR (escape code 1) are not stored in the metafile.
- The effect GpiErase, including close-segment processing, is recorded in a metafile.
- The following functions can produce unexpected effects if the metafile contents are displayed on a different device from the one they were created for:
 - GpiBitBlt
 - GpiSetPel
 - GpiSetClipRegion
 - GpiOffsetClipRegion
 - GpiPaintRegion
 - GpiIntersectClipRectangle
 - GpiExcludeClipRectangle.

The reason for the unexpected effects is because the pel resolutions of the devices might differ. Raster operations (for example, GpiBitBlt) do not work on plotters.

- You can associate the metafile device context with a different presentation space while creating the metafile, but the new presentation space must have the same format as the original.

Creating Metafiles for Interchange: The following restrictions apply if the metafile is to be played when the current drawing-mode parameter is DM_DRAWANDRETAIN or DM_RETAIN, or if the metafile is to conform to SAA guidelines:

- Before you issue the first drawing instruction to a graphics presentation space that has been associated with a metafile device context, you must establish (or default) the following:
 - The graphics field (use GpiSetGraphicsField)
 - The color table (use GpiCreateLogColorTable)
 - The code page for the default character set (use GpiSetCp)
 - The default viewing transformation (use GpiSetDefaultViewMatrix)
 - The settings of the drawing controls (use GpiSetDrawControl; DCTL_DISPLAY must be set on)
 - The default values of attributes (use GpiSetDefAttrs)
 - The default viewing limits (use GpiSetDefViewingLimits)
 - The primitive tag (use GpiSetDefTag)
 - The default arc parameters (use GpiSetDefArcParams).

You must not specify a graphics field if the metafile is to conform to SAA guidelines. If a graphics field is specified, you can play the metafile in DM_RETAIN or DM_DRAWANDRETAIN mode, but the graphics field must be specified before the first drawing instruction is issued to the metafile. The effect of the graphics field in the target presentation space when playing the metafile is controlled by the PMF_LOADTYPE option of GpiPlayMetafile.

- You can define logical fonts and identify bit maps to be used as area-fill patterns at any time. You must not, however, issue GpiDeleteSetId against these resources after they have been established. Once assigned, therefore, lcid's cannot be reused.
- The size of the logical-color-table data must not exceed 31KB.
- You must not use clipping regions. Therefore, none of the following functions is supported:
 - GpiSetClipRegion
 - GpiExcludeClipRectangle
 - GpiIntersectClipRectangle
 - GpiOffsetClipRegion.
- You must not reassociate the metafile device context.
- When you use a bit map as the source of a GpiWCBtBlit operation or as an area-fill pattern, it must not be modified.
- Only the following foreground mixes can be used:
 - FM_DEFAULT
 - FM_OR
 - FM_OVERPAINT
 - FM_LEAVEALONE.
- Only the following background mixes can be used:
 - BM_DEFAULT
 - BM_OVERPAINT
 - BM_LEAVEALONE.

- The following functions are not supported:

- GpiBitBlt
- GpiSetPel
- GpiSetPS
- GpiResetPS
- GpiErase
- GpiPaintRegion
- DevEscape.

Micro Presentation Space Restrictions: When you create a metafile from a micro presentation space, or play the metafile through a micro presentation space, the contents of the metafile are restricted to those GPI functions that are valid in a micro presentation space. In this case, the application must use GpiDestroyPS instead of GpiAssociate before issuing DevCloseDC. GpiAssociate is not permitted in a micro presentation space.

Query Restrictions: Query functions can be issued, with the usual restrictions, unless the metafile device context was created using the no query option, OD_METAFILE_NOQUERY.

Segment Restrictions: Chained and unchained segments invoked by any GpiDraw command are written to the metafile. Primitives outside segments are recorded automatically as zero segments. If the metafile is subsequently played in DM_RETAIN mode, all graphics are directed to the segment store.

Chained dynamic segments cannot be recorded in a metafile. GpiRemoveDynamics and GpiDrawDynamics cause an error condition to be raised when the presentation space is associated with a metafile device context. Unchained dynamic segments are recorded as zero segments. To draw an unchained dynamic segment, use GpiDrawSegment.

Asynchronous Drawing Thread Restrictions: If the metafile is being created on an asynchronous drawing thread and the thread is suspended by GpiSetStopDraw, an unusable metafile results.

Metafile Functions

OS/2 2.0 provides a set of functions that allow you to:

- Create a metafile
- Store pictures in a metafile
- Play a metafile
- Save a metafile
- Load a metafile
- Edit a metafile
- Copy a metafile
- Delete a metafile.

How these functions are used to create and manipulate metafiles in relationship to applications and components of the operating system is illustrated in Figure 15-1.

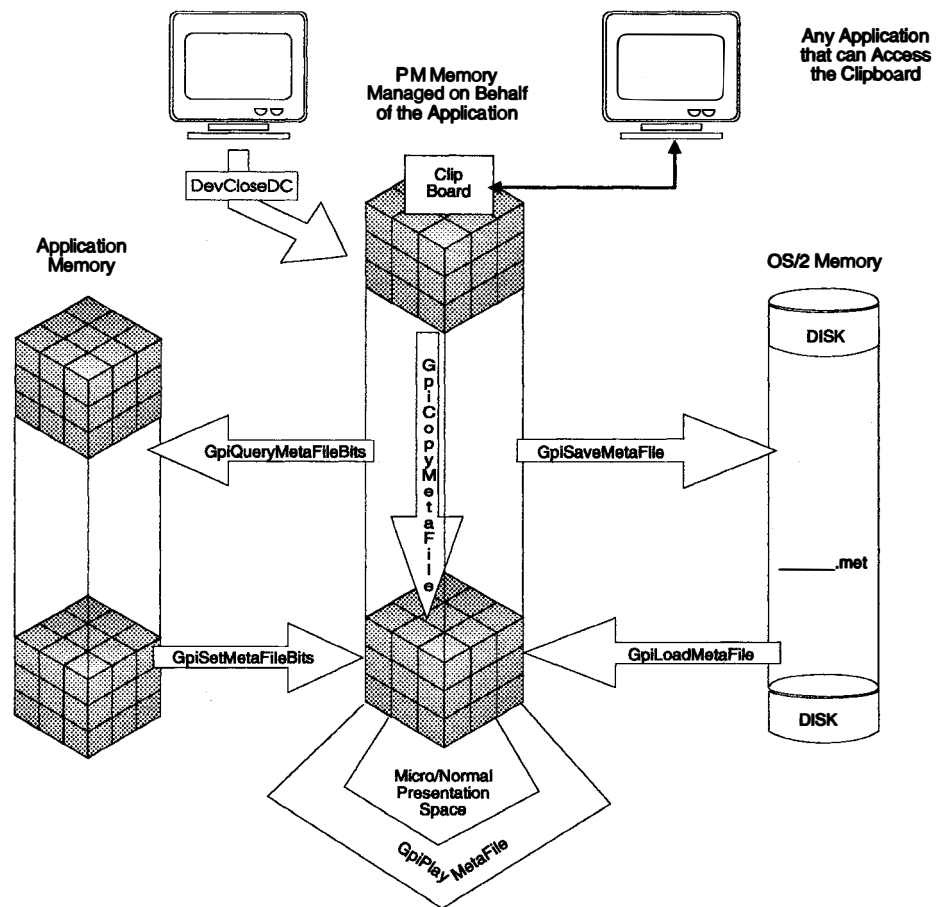


Figure 15-1. Metafile Functions

Creating a Metafile

Pictures are drawn in presentation spaces associated with device contexts. PM considers a metafile to be an output device or destination, in the same manner as a window or printer. This means that to record a picture in a metafile, a metafile device context must be created and associated with a presentation space.

A metafile device context is created by calling `DevOpenDC` and specifying the type of device context as:

- `OD_METAFILE`
- `OD_METAFILE_NOQUERY`.

`OD_METAFILE` is generally used to specify a metafile device context, although `OD_METAFILE_NOQUERY` provides better recording performance. `OD_METAFILE_NOQUERY` does not support attribute queries, for example, `GpiQueryCurrentPosition`, or `GpiQueryColor`.

The device driver for the device that the metafile device context is associated with is specified in the DEVOPENSTRUC data structure required for DevOpenDC. DEVOPENSTRUC is described in Chapter 18, "Print Job Submission and Manipulation."

A metafile device context can be associated with a newly-created presentation space by calling GpiCreatePS and specifying the handle to the metafile device context returned from the call to DevOpenDC. A metafile device context can also be associated with an existing, normal presentation space by calling GpiAssociate and specifying the handle to the metafile device context returned from the call to DevOpenDC.

Storing Pictures in a Metafile

When a metafile device context is associated with a presentation space, presentation space resources (such as the current logical color table) and environmental settings (such as the presentation-page format) are copied automatically into the metafile. These items must be established *before* the presentation space is associated with the metafile device context.

Loading of additional resources (such as fonts) and adjustments to the environment (such as modifying the default viewing transform) should be made before you begin drawing. Attribute settings, segment-creation requests, and primitive-drawing requests that contribute to the picture are directed to the metafile *after* its device context has been associated with a presentation space.

If an application calls GpiSetDrawControl, specifying DCTL_DISPLAY and DCTL_OFF, before drawing graphics into a metafile, the graphics are not visible when the metafile is played. However, the metafile records any changes made to the current position or presentation-space attributes.

When the metafile device context has been associated with a graphics presentation space, the metafile is ready to receive graphical data. Just as with any other output destination, whether the picture is sent directly to the metafile is controlled by the current drawing mode, as shown in Table 15-1.

<i>Table 15-1. Drawing Mode Dependencies When Recording Metafiles</i>	
Drawing Mode	Effect
Draw mode	Graphics go directly to the metafile.
Retain mode	Graphics go to the segment store of the presentation space. They are not directed to the metafile until the application issues an appropriate GpiDraw request (GpiDrawChain, GpiDrawFrom, GpiDrawSegment).
Draw-and-retain mode	Graphics go directly to the metafile, and also to the segment store of the presentation space.

The drawing mode can be changed at any time while the metafile device context remains open by calling GpiSetDrawingMode.

As long as the metafile device context remains open, you can continue drawing. A metafile can only contain data generated from GDI functions. Any nongraphical data included in a metafile is ignored. The following list describes items found in a metafile:

- Picture
- Logical color table
- Logical font
- Fill pattern
- Viewing transformation
- Page units
- Page dimensions.

When an application finishes drawing in a metafile, it must disassociate the metafile device context from the presentation space by calling `GdiAssociate`. If the metafile is associated with or through a micro presentation space, call `GdiDestroyPS` to perform an implicit disassociation.

When you have finished drawing in the metafile, and the presentation space has been disassociated, the application can close the metafile device context and obtain a handle to the metafile by calling `DevCloseDC`. A closed metafile cannot be reopened; therefore, additional drawing in the metafile is not possible. A closed metafile can be referenced by the *metafile handle*. The metafile handle is used to reference the metafile for subsequent operations on the metafile. Use the metafile handle to:

- Transfer a metafile to application memory
- Transfer a metafile from application memory
- Save a metafile on disk
- Play a metafile into presentation space
- Edit a metafile
- Copy a metafile
- Delete a metafile.

Because each metafile can be distinctly identified, your application can work with more than one metafile at a time. However, because metafiles can be very large files, you must make maximum use of the metafile handles to avoid duplicating the actual metafiles in memory.

Playing a Metafile

An application can redraw a picture stored in a metafile by executing the contents of metafile. This is known as *playing* a metafile into a graphics presentation space. How a picture is redrawn (that is, how the metafile is replayed) depends on the current drawing-mode of the target presentation space. Table 15-2 describes these dependencies.

Table 15-2. Drawing Mode Dependencies When Playing Metafiles		
Drawing Mode	Result	If the metafile contains a segment chain...
DM_DRAW	The metafile contents are executed, and the picture is directed to the current output device.	The contents of the segments in the chain represent the picture output that is directed at the target output device.
DM_RETAIN	The metafile contents are retained in the application's segment store. The picture defined in the metafile is not directed to an output device until the application issues an appropriate GpiDraw.	The chain is added to the end of any chain that may already be in the segment store. If any segment in the metafile has the same nonzero name as a segment already in the segment store of the presentation space, an error condition is raised.
DM_DRAWANDRETAIN	The metafile contents are both stored and executed.	
Note: Unchained segments in the metafile are always retained, regardless of the current drawing-mode parameter.		

When the contents of a metafile are retained in the segment store of the target presentation space, they can be manipulated by the application as if the application had created them. For example, the segments can be edited, drawn, correlated, and deleted.

A metafile is "played" by calling GpiPlayMetaFile. A metafile cannot, however, be "played" within a segment bracket. GpiPlayMetaFile requires a metafile handle, an option count, an options array, a byte count of a descriptive record, and the descriptive record.

The GpiPlayMetaFile descriptive record is a natural-language description of the picture contents. For example, its value might be *A House*. This description is provided on input to the DevOpenDC function that defines the metafile device context. It can be used, for example, in a list box from which a user can select a picture.

The GpiPlayMetaFile options array specifies how the operating system alters your application's presentation space before playing the metafile. The options array determines the display attributes of the metafile. The array fields and their respective attributes, as numbered consecutively in the array, are shown in Table 15-3.

<i>Table 15-3. GpiPlayMetaFile Options</i>	
Option	Attribute
PMF_SEGBASE	Reserved, must be 0.
PMF_LOADTYPE	Viewing transform, graphics.
PMF_RESOLVE	Reserved, must be 0.
PMF_LCIDS	Font and local bit map local identifiers.
PMF_RESET	Drawing attributes.
PMF_SUPPRESS	Actual playing of the metafile.
PMF_COLORTABLES	Color tables, color palettes.
PMF_COLORREALIZABLE	Realization of metafile color table.
PMF_DEFAULTS	Drawing defaults values.

PMF_RESET Option: The PMF_RESET option, more so than the other array options, controls the effects of other array options. This option allows a total reset of the presentation space and provides further control of the following presentation space environmental attributes:

- Page units (device transform)
- Page dimensions
- Default viewing transform.

Warning: If the metafile page units, page dimensions and page coordinate format do not match those in the presentation space, playing the metafile will cause an error.

The PMF_RESET option can have one of the values shown in Table 15-4.

Table 15-4. PMF_RESET Options for GpiPlayMetaFile	
Value	Description
RES_RESET	Allows the presentation space to be completely reset, just as if it were newly created, and the values of the aforementioned environmental attributes to be specified from the metafile into the presentation space. This option is equivalent to specifying GRES_ALL on GpiResetPS. If this option is used, the target presentation page is redefined so that it is the same size, and defined in the same units, as the metafile. It also ensures that the physical size of the metafile picture is preserved if the presentation page is defined in units other than pels, or arbitrary units.
RES_NORESET	Prevents resetting of the presentation space. The existing presentation space values of the environmental attributes will be preserved, and their values in the metafile will be discarded. If this option is used, ensure that the presentation-page units of the target presentation space are the same as those with which the metafile was generated, before you play a metafile into the presentation space. For example, if the original presentation page is defined in increments of 0.1 mm, the target presentation page must be defined in the same way. To change the format of the target presentation page, if the presentation-page formats do not match, issue GpiSetPS, but only if you know the metafile presentation-page units.
RES_DEFAULT	

If you are certain that the settings of the current presentation space are appropriate for the metafile, then you can play the metafile with the NO_RESET option. To reset some or all of the current values in the target presentation space, without redefining the presentation page, call:

1. GpiResetPS, with the GRES_ATTRS or GRES_SEGMENTS option
2. GpiPlayMetaFile, with the NO_RESET option.

The PMF_LCID option may be used to append or replace fonts from the metafile to the fonts in the presentation space. The PMF_COLORTABLES option may be used to append or replace color table entries in the presentation space from the metafile. Other options (PMF_LOADTYPE, PMF_DEFAULTS, and so on) can be used to selectively replace settings in the presentation space from the metafile.

Like the PMF_RESET option, other options, except for PMF_SUPPRESS, either:

- Allow the values in the presentation space to be preserved and those in the metafile to be discarded, or
- Allow the presentation space to be loaded using the values in the metafile.

If the first choice is used with RES_RESET, then the items controlled by the PMF_options will be left set to their default reset state, otherwise it will occur as described in the first choice above.

If the second choice is used with RES_RESET, then the items controlled by the PMF_options are as described above. If it is used with RES_NORESET, then the fonts and color tables in the metafile can append to the existing color tables and fonts in the presentation space. Existing presentation space color table entries and fonts, such as viewing transform will be replaced if respecified in the metafile.

Note: If your application uses the attributes from the metafile, the attributes that were specified in the presentation space before the metafile was played are overwritten.

PMF_SUPPRESS Option: The PMF_SUPPRESS option allows you to control the playing of the metafile. The PMF_SUPPRESS option can have one of the values shown in Table 15-5.

Table 15-5. PMF_SUPPRESS Options for GpiPlayMetafile	
Value	Description
SUP_SUPPRESS	The operating system does not draw the metafile on the device associated with the presentation space. This option is useful if you need to alter the presentation space before drawing it on an output device.
SUP_NOSUPPRESS SUP_DEFAULT	The operating system draws the metafile on the device associated with the presentation space.

You can use the RES_RESET option to reinitialize the presentation space and modify certain resources or presentation space environmental attributes from the application before playing the remainder of the metafile by:

1. Saving the current state of the presentation space by calling GpiSavePS.

Use this function only if you wish to restore the previous state of the presentation space after playing the metafile.

2. Calling GpiPlayMetaFile and specifying:

- The PMF_RESET option with the value RES_RESET
- The PMF_SUPPRESS option with the value SUP_SUPPRESS.

This causes GpiPlayMetaFile to set the presentation space to the specifications recorded in the metafile but not to play the picture in the metafile.

3. Making any required changes to the presentation space.

4. Calling GpiPlayMetaFile and specifying:

- The PMF_RESET option with the value RES_NORESET.
- The PMF_SUPPRESS option with the value SUP_NOSUPPRESS.

This causes the picture to play.

5. Restoring the state of the presentation space after playing the metafile by calling GpiRestorePS.

PMF_LOADTYPE Option: The PMF_LOADTYPE option of GpiPlayMetaFile determines which viewing transformations and graphics fields affect the metafile picture. The PMF_LOADTYPE option can have one of the values shown in Table 15-6.

Table 15-6. PMF_LOADTYPE Options for GpiPlayMetafile	
Value	Description
LT_ORIGINALVIEW	Viewing transformations recorded in the metafile are used in the target presentation space. If changes to the graphics field are recorded in the metafile, the graphics field in the target presentation space is updated, and the picture is clipped to that graphics field. Changes to the default viewing transformation recorded in the metafile are used to update the default viewing transformation in the target presentation space.
LT_NOMODIFY LT_DEFAULT	Any viewing transformations recorded in the metafile are ignored. The metafile contents are drawn in accordance with the current viewing transformation in the target presentation space. (Note that, if you also specify RES_RESET, the current viewing transformation is reset to its default value.) The picture is clipped to any graphics field defined in the target presentation space, and any graphics field recorded in the metafile is ignored.

The PMF_RESET option RES_RESET changes values in the target presentation space. Therefore, the effect of the PMF_LOADTYPE option should always be considered in conjunction with the PMF_RESET option. For example, if you specify both RES_RESET and LT_NOMODIFY, the default viewing transformation in the target presentation space is updated with that from the metafile, even though LT_NOMODIFY means that such values should be ignored.

PMF_LCIDS Option: The PMF_LCIDS option of GpiPlayMetaFile controls whether logical resources referenced by a local identifier (lcid) are loaded from the metafile. Logical resources include logical fonts and bit maps used as area-fill patterns. The PMF_LCIDS option can have one of the values shown in Table 15-7.

Table 15-7. PMF_LCIDS Options for GpiPlayMetafile	
Value	Description
LC_LOADDISC	Loads lcid-referenced resources from the metafile. If the lcids of those resources are already in use in the target presentation space, the resources currently identified by those lcids are deleted and their lcids are freed before the metafile contents are loaded. The new fonts and bit maps replace the existing ones in the presentation space. (If the operating system uses a local identifier that the application has already defined, GpiPlayMetaFile deletes the existing identifier before using it for the metafile resource.)
LC_NOLOAD LC_DEFAULTS	The operating system uses the presentation space's logical font and custom fill pattern; it will ignore any logical font or custom fill pattern in the metafile. An application can use the GpiSavePS and GpiRestorePS functions to maintain the local identifiers it has already defined.

PMF_COLORTABLE Option: The PMF_COLORTABLES option may be used to append or replace color table entries in the presentation space from the metafile. The PMF_COLORTABLES option can have one of the values shown in Table 15-8.

Table 15-8. PMF_COLORTABLES Options for GpiPlayMetafile	
Value	Description
CTAB_REPLACE	Replaces the logical color table in the presentation space with the color table in the metafile.
CTAB_REPLACEPALETTE	Replaces the current palette, if it exists, in the presentation space with the palette in the metafile.
CTAB_NOMODIFY CTAB_DEFAULT	Maintains the logical color table in the presentation space.

PMF_COLORREALIZABLE Option: The PMF_COLORREALIZABLE option may be used to select whether color table from the metafile is realized upon loading. The PMF_COLORREALIZEABLE option can have one of the values shown in Table 15-9.

Table 15-9. PMF_COLORREALIZABLE Options for GpiPlayMetafile	
Value	Description
CREA_DOREALIZE	Sets the realizable option when loading the color table.
CREA_NOREALIZE CREA_DEFAULT	Does not set the realizable option when loading the color table.

PMF_DEFAULTS Option: The PMF_DEFAULTS option can have one of the values shown in Table 15-10.

Table 15-10. PMF_DEFAULTS Options for GpiPlayMetafile	
Value	Description
DDEF_LOADDISC	Replaces the default attributes, default viewing limits, and default arc parameters in the presentation space with the values specified in the metafile.
DDEF_IGNORE DDEF_DEFAULTS	Maintains the default attributes, default viewing limits, and default arc parameters in the presentation space.

Saving a Metafile

A metafile is not a permanent object and, unless it is explicitly saved, disappears when the application that creates it ends. Metafiles are saved in disk files by calling GpiSaveMetaFile.

This function accepts as input the metafile handle and the name of the disk file in which the metafile is to be saved. As the disk file is created by this function, an error condition is raised if a file with this name already exists. When you call GpiSaveMetaFile, the memory version of the metafile is deleted. Its handle is no longer valid, until the file is reloaded from disk.

Loading a Metafile

A metafile saved on disk has to be reloaded before it can be used. Any application with access to the disk file can transfer the file from disk storage to application storage by calling GpiLoadMetaFile. GpiLoadMetaFile returns a metafile handle, which identifies the metafile after it has been loaded.

Editing a Metafile

Metafiles can be edited in a manner similar to editing graphics segments. To edit graphics orders in a metafile, an application transfers them into application storage (an array of bytes) by calling the GpiQueryMetaFileBits function. The number of graphics orders copied depends on the size of the array that the application supplies. The application can determine the size of the metafile (in bytes) by calling the GpiQueryMetaFileLength function. When the application has finished editing the graphics orders, it can copy them back into the metafile by calling the GpiSetMetaFileBits function.

Edited versions of metafiles can be saved by calling GpiSaveMetaFile. The disk file that contains the edited version of the metafile cannot have the same name as the file from which the metafile was originally loaded. GpiSaveMetaFile raises an error condition if the disk file already exists.

Copying a Metafile

A copy of a memory metafile can be made by calling GpiCopyMetaFile. PM makes a copy of the metafile and returns a new handle to the new metafile.

Deleting a Metafile

A memory metafile is deleted by calling `GpiDeleteMetaFile`. After the operation is complete, the handle to the metafile no longer points to a usable object. A disk file containing the metafile remains untouched by this operation. A disk metafile is deleted by using operating system commands that delete any other type of file.

Metafiles and the System Clipboard

A metafile that you have created or loaded can be made available to other applications at the same workstation by placing the handle to the metafile in the system clipboard. When a resource is passed to the clipboard, it becomes the property of the system, and is not deleted when the owning application ends. Any application that can access the clipboard can retrieve the handle to the metafile. Using the handle to the metafile, the application should copy the metafile by calling `GpiCopyMetaFile` before closing the clipboard. The application can then perform any metafile operation on the metafile.

For more information on the system clipboard, see the *Programming Guide Volume II*.

Using Metafiles

Use metafile functions to:

- Create a metafile
- Draw into a metafile
- Load a metafile to disk
- Load a metafile from disk into an application
- Play a metafile
- Edit a metafile
- Copy a metafile
- Transfer metafile contents to application memory
- Transfer metafile contents from application memory.

Creating and Drawing into a Metafile

To create a metafile, you must:

1. Create a metafile device context with `DevOpenDC`.
2. Create a presentation space with `GpiCreatePS` function, and associate the presentation space with the metafile device context.
3. Draw into the metafile with various Gpi drawing functions.
4. Disassociate the metafile device context from the presentation space with `GpiAssociate`.
5. Close the metafile device context with `DevCloseDC`.

Figure 15-2 shows how to create a simple metafile that draws text within the borders of a three-color box.

```
#include <os2.h>
void fncMETA01(void){
    DEVOPENSTRUC dop;
    HDC hdcMeta;
    HPS hpsMeta;
    HMF hmf;
    HAB hab;
    SIZEL sizlPage;
    POINTL ptl;

    dop.pszLogAddress = (PSZ) NULL;
    dop.pszDriverName = "DISPLAY";

    hdcMeta = DevOpenDC(hab,
        OD_METAFILE,           /* Metafile device context */
        "*",                   /* Ignores OS2.INI */
        2L,                    /* Uses first two fields */
        (PDEVOPENDATA) &dop,    /* Device information */
        (HDC) NULLHANDLE);     /* Compatible device context */

    hpsMeta = GpiCreatePS(hab,
        hdcMeta,               /* Metafile device context */
        &sizlPage,              /* Page viewport */
        PU_PELS | GPIA_ASSOC); /* Device units and associated context */

    /* Draw a box in a metafile. */

    GpiSetColor(hpsMeta, CLR_CYAN);

    ptl.x = 150; ptl.y = 200;
    GpiMove(hpsMeta, &ptl);

    ptl.x = 300; ptl.y = 275;
    GpiBox(hpsMeta, DRO_FILL, &ptl, 0L, 0L);

    GpiSetColor(hpsMeta, CLR_GREEN);

    ptl.x = 300; ptl.y = 200;
    GpiMove(hpsMeta, &ptl);

    ptl.x = 390; ptl.y = 275;
    GpiBox(hpsMeta, DRO_FILL, &ptl, 0L, 0L);

    GpiSetColor(hpsMeta, CLR_YELLOW);

    ptl.x = 390; ptl.y = 200;
    GpiMove(hpsMeta, &ptl);

    ptl.x = 530; ptl.y = 275;
    GpiBox(hpsMeta, DRO_FILL, &ptl, 0L, 0L);

    ptl.x = 175; ptl.y = 230;
    GpiMove(hpsMeta, &ptl);
```

Figure 15-2 (Part 1 of 2). Creating a Metafile

```

    GpiSetColor(hpsMeta, CLR_PINK);

    GpiCharString(hpsMeta, 41,
        "METAFILE COPY METAFILE COPY METAFILE COPY");

    GpiAssociate(hpsMeta, (HDC) NULLHANDLE);
    hmf = DevCloseDC(hdcMeta);
} /* fncMETA01 */

```

Figure 15-2 (Part 2 of 2). Creating a Metafile

Drawing into a Metafile in Retain Mode

To draw into a metafile, set the drawing mode to the appropriate value for your application and then perform the drawing operations. Figure 15-3 shows an example of how to copy the contents of a segment into a metafile.

```

#define INCL_GPIMETAFILES
#define INCL_GPICONTROL
#include <os2.h>
void fncMETA02(void){
    HDC hdcMeta, hdc;
    POINTL ptl;
    HMF hmf;
    LONG alOpt[10];
    HPS hps;

    /*
     * Open a segment, assign it an identifier of 10,
     * and draw some text into it.
     */

    GpiSetDrawingMode(hps, DM_RETAIN);
    GpiOpenSegment(hps, 10L);
    ptl.x = 175; ptl.y = 230;
    GpiMove(hps, &ptl);
    GpiSetColor(hps, CLR_PINK);
    GpiCharString(hps, 41,
        "METAFILE COPY METAFILE COPY METAFILE COPY");
    GpiCloseSegment(hps);

    GpiAssociate(hps, NULLHANDLE); /* Disassociates PS and screen DC */
    GpiAssociate(hps, hdcMeta);    /* Associates PS and meta DC */
    GpiDrawSegment(hps, 10L);      /* Draws segment into metafile */
    GpiAssociate(hps, NULLHANDLE); /* Disassociates PS and meta DC */
    hmf = DevCloseDC(hdcMeta);     /* Closes metafile */

    GpiAssociate(hps, hdc);        /* Associates PS and screen DC */
    GpiSetDrawingMode(hps, DM_DRAW); /* Sets drawing mode to DM_DRAW */

    /*
     * Load the array of options for GpiPlayMetaFile
     * with default values.
     */
}

```

Figure 15-3 (Part 1 of 2). Copying a Graphic Segment to a Metafile


```

a1Opt[PMF_SEGBASE] = 0; /* Reserved */
a1Opt[PMF_LOADTYPE] = LT_DEFAULT; /* Default transformations */
a1Opt[PMF_RESOLVE] = 0; /* Reserved */
a1Opt[PMF_LCIDS] = LC_DEFAULT; /* Uses default lcids */
a1Opt[PMF_RESET] = RES_DEFAULT; /* Uses default */
a1Opt[PMF_SUPPRESS] = SUP_DEFAULT; /* Uses default */
a1Opt[PMF_COLORTABLES] = CTAB_DEFAULT; /* Uses default */
a1Opt[PMF_COLORREALIZABLE] = CREA_DEFAULT; /* Uses default */

GpiPlayMetaFile(hps, /* Plays metafile onto screen */
hmf, 8L, a1Opt, (PLONG) NULL, 0L, (PSZ) NULL);
} /* fncMETA02 */

```

Figure 15-3 (Part 2 of 2). Copying a Graphic Segment to a Metafile

If you want to create a simple drawing in a metafile for repeated display, you can set the drawing mode to `DM_DRAW` and draw directly into the metafile. The code in the first code fragment shows an example of how to do this. You can store named segments in the metafile that you are recording in `DM_DRAW` mode by bracketing your output primitive functions between `GpiOpenSegment` and `GpiCloseSegment`.

Copying a Metafile to Disk

You can copy a metafile to disk using `GpiSaveMetaFile` function, and you can load the file back into your application using `GpiLoadMetaFile` function. Figure 15-4 shows an example of how to copy a metafile to a file named `META.MET`, then load the same file back into the application and play it.

```

#define INCL_GPIMETAFILES
#include <os2.h>
void fncMETA03(void){
    HMF hmf;
    HAB hab;
    HPS hps;
    LONG a1Opt[10];

    GpiSaveMetaFile(hmf, "meta.met"); /* Saves metafile on disk */

    hmf = GpiLoadMetaFile(hab, "meta.met"); /* Loads metafile */

    a1Opt[PMF_SEGBASE] = 0; /* Reserved */
    a1Opt[PMF_LOADTYPE] = LT_DEFAULT; /* Default transformations */
    a1Opt[PMF_RESOLVE] = 0; /* Reserved */
    a1Opt[PMF_LCIDS] = LC_DEFAULT; /* Uses default lcids */
    a1Opt[PMF_RESET] = RES_DEFAULT; /* Uses default */
    a1Opt[PMF_SUPPRESS] = SUP_DEFAULT; /* Uses default */
    a1Opt[PMF_COLORTABLES] = CTAB_DEFAULT; /* Uses default */
    a1Opt[PMF_COLORREALIZABLE] = CREA_DEFAULT; /* Uses default */

    GpiPlayMetaFile(hps, hmf, 8, a1Opt, (PLONG) NULL, 0L, (PSZ) NULL);
} /* fncMETA03 */

```

Figure 15-4. Copying a Metafile to Disk

Playing a Metafile

Figure 15-5 shows an example of how to play the metafile using the font, color table, and fill-pattern descriptions in your application's presentation space.

```
#define INCL_GPIMETAFILES
#include <os2.h>
void fncMETA04(void){
    HPS hps;
    HMF hmf;
    LONG a1Opt[10];

    a1Opt[PMF_SEGBASE] = 0;           /* Reserved */
    a1Opt[PMF_LOADTYPE] = LT_DEFAULT; /* Viewing transformation in PS */
    a1Opt[PMF_RESOLVE] = RS_DEFAULT;  /* Reserved */
    a1Opt[PMF_LCIDS] = LC_DEFAULT;    /* Font and fill pattern in PS */
    a1Opt[PMF_RESET] = RES_DEFAULT;   /* Page units and dimensions in PS */
    a1Opt[PMF_SUPPRESS] = SUP_DEFAULT; /* Draws metafile into PS */
    a1Opt[PMF_COLORTABLES] = CTAB_DEFAULT; /* Color table in PS */
    a1Opt[PMF_COLORREALIZABLE] = CREA_DEFAULT; /* Sets realizable option */

    GpiPlayMetaFile(hps, hmf, 8, a1Opt, (PLONG) NULL, 0L, (PSZ) NULL);
} /* fncMETA04 */
```

Figure 15-5. Playing a Metafile

Figure 15-6 shows an example of how to play a metafile using the font, color table, and fill-pattern descriptions in the metafile.

```
#define INCL_GPIMETAFILES
#include <os2.h>
void fncMETA05(void){
    HPS hps;
    HMF hmf;
    LONG a1Opt[10];

    a1Opt[PMF_SEGBASE] = 0;           /* Reserved */
    a1Opt[PMF_LOADTYPE] = LT_DEFAULT; /* Viewing transformation in PS */
    a1Opt[PMF_RESOLVE] = RS_DEFAULT;  /* Reserved */
    a1Opt[PMF_LCIDS] = LC_LOADDISC;   /* Font and fill pattern in metafile */
    a1Opt[PMF_RESET] = RES_DEFAULT;   /* Page units and dimensions in PS */
    a1Opt[PMF_SUPPRESS] = SUP_DEFAULT; /* Draws metafile into PS */
    a1Opt[PMF_COLORTABLES] = CTAB_REPLACE; /* Color table in metafile */
    a1Opt[PMF_COLORREALIZABLE] = CREA_DEFAULT; /* Sets realizable option */

    GpiPlayMetaFile(hps, hmf, 8, a1Opt, (PLONG) NULL, 0L, (PSZ) NULL);
} /* fncMETA05 */
```

Figure 15-6. Playing a Metafile Using Font, Color, and Fill-Pattern Descriptions

Figure 15-7 shows an example of how to play a metafile using the viewing transformation, page units, and presentation-page dimensions specified in the metafile.

```
#define INCL_GPIMETAFILES
#include <os2.h>
void fncMETA06(void){
    HPS hps;
    HMF hmf;
    LONG a1Opt[10];

    a1Opt[PMF_SEGBASE] = 0;                /* Reserved */
    a1Opt[PMF_LOADTYPE] = LT_ORIGINALVIEW; /* Viewing transformation */
    a1Opt[PMF_RESOLVE] = RS_DEFAULT;       /* Reserved */
    a1Opt[PMF_LCIDS] = LC_DEFAULT;         /* Font and fill pattern in PS */
    a1Opt[PMF_RESET] = RES_RESET;          /* Page units/dimensions */
    a1Opt[PMF_SUPPRESS] = SUP_DEFAULT;     /* Draws metafile into PS */
    a1Opt[PMF_COLORTABLES] = CTAB_DEFAULT; /* Uses color table in PS */
    a1Opt[PMF_COLORREALIZABLE] = CREA_DEFAULT; /* Sets realizable option */

    GpiPlayMetaFile(hps, hmf, 8, a1Opt, (PLONG) NULL, 0L, (PSZ) NULL);
} /* fncMETA06 */
```

Figure 15-7. Playing a Metafile Using Transformation and Page Units

Summary

Table 15-11 summarizes the metafile object functions.

<i>Table 15-11. Metafile Functions</i>	
Function Name	Description
DevCloseDC	Closes a metafile device context and returns a handle that identifies a metafile.
DevOpenDC	Creates a metafile device context when you pass it the OD_METAFILE constant as the second argument.
GpiCopyMetaFile	Creates a copy of a metafile.
GpiDeleteMetaFile	Deletes a metafile.
GpiLoadMetaFile	Loads data from disk storage into a metafile and returns a handle that identifies the metafile.
GpiPlayMetaFile	Plays the contents of a metafile into a presentation space.
GpiQueryMetaFileBits	Copies the contents of a metafile (such as drawing orders, color-table description, and page units) into an array of bytes. The number of bytes copied depends on the size of the array.
GpiQueryMetaFileLength	Retrieves the size of a metafile in bytes. You use this function to determine the size of the array you will need when you issue the GpiQueryMetaFileBits function.
GpiSaveMetaFile	Copies a metafile to a disk and then removes it from your application's memory.
GpiSetMetaFileBits	Copies drawing orders from an array of bytes to a metafile. You use this function to copy drawing orders back into a metafile from application memory.

Chapter 16. Clipping and Boundary Determination

Both Clipping and boundary determination are graphics operations concerned with limiting the amount of graphics information passed between different coordinate spaces. The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Coordinate spaces
- Transformations
- Regions
- Paths.

About Clipping

Clipping enables the PM programming interface to discard parts of a picture that lie outside a specified *clipping boundary*. The parts of the picture enclosed by the boundary are said to be inside the *clipping area*. A clipping area might be a single rectangle or a complex shape, depending on the method used to define it.

Note: In this chapter, the word *area* does not refer to an *area primitive*; it describes the shape or shapes used for clipping graphics output.

If an application attempts to draw outside of a clipping area, the operating system ensures that the output does not appear on the drawing surface of the output device. For example, if an application defined a triangular clipping area before drawing text output, all text outside of the triangle would be discarded, even though an entire page of text was defined. Figure 16-1 illustrates the result.

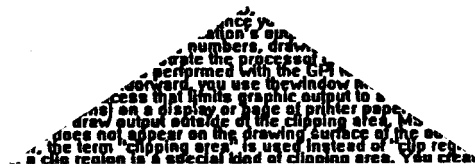


Figure 16-1. Triangular Clip Path

Clipping boundaries are defined by the application. The PM programming interface performs some clipping automatically when, for example, your application's graphic output is clipped to fit a client window area or the device's output area for a hardcopy device.

Types of Clipping Areas

Clipping can occur at each stage in the *viewing pipeline*. The viewing pipeline is described and illustrated in Chapter 17, "Coordinate Spaces and Transformations." Objects in world coordinate, model, page, or device space can be clipped. When an application defines clipping areas in several coordinate spaces, the final result is similar to combining all the areas into a single clipping area. This single area is defined by the intersection of the areas in each coordinate space.

Clipping in different coordinate spaces, however, is a means of *conceptualizing* the process to aid in its understanding. Clipping, like transformations, actually happens in one operation, with all the different types of clipping being performed on all primitives in the device space at once.

Table 16-1 describes the different types of clipping areas associated with the different coordinate spaces.

Table 16-1. Clipping Areas and Coordinate Space Summary		
Clipping Area	Coordinate Space	Description
Clip path	World space	• Always inclusive/inclusive • Clipping area can have curved edges • Clipping area can be rotated.
Viewing limit	Model space	• Always a rectangular clipping boundary • Always inclusive/inclusive • Rotating clipping area results in larger rectangle.
Graphics field	Page space	• Always a rectangular clipping boundary • Always inclusive/inclusive • Rotating clipping area impossible. Cannot specify a device transform with rotation.
Clip region	Device space	• Can be a single rectangle or multiple rectangles that overlap or remain separate • Always inclusive/exclusive • Rotating the clipping area is impossible.
Note: <i>Inclusive/inclusive</i> means that the operating system includes the bottom and leftmost edges of the rectangle in the clipping area as well as the top and rightmost edges. <i>Inclusive/exclusive</i> means that the operating system includes the bottom and leftmost edges but excludes the top and rightmost edges. This concept is illustrated in Figure 16-2 on page 16-3		

Figure 16-2 illustrates the differences between inclusive-inclusive and inclusive-exclusive clipping.

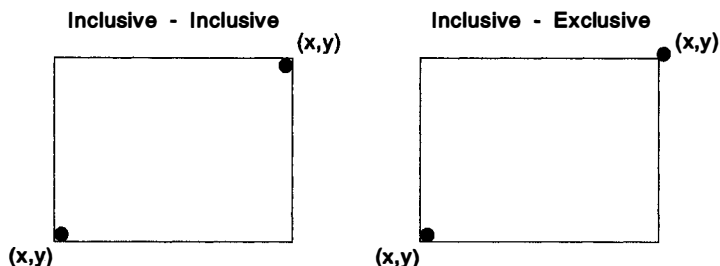


Figure 16-2. Inclusive/Inclusive and Inclusive/Exclusive Clipping

The Clip Path

The *clip path* defines the clipping boundary in world coordinate space. Paths are graphic objects that serve many purposes, only one of which is to be used as a clipping mechanism. Paths are described in Chapter 10, "Paths." The recommended functional sequence for creating a clip path is as follows:

Function	Effect
GpiBeginPath	Begins the path definition.
Line and arc GpIs	Gives the path shape.
GpiEndPath	Ends the path definition.
GpiModifyPath	An optional step explained below.
GpiSetClipPath	Converts the path to a clip path.

Before converting to a clip path, the path can be modified using `GpiModifyPath`. If modified, the path is converted to a geometric (wide) line using the current geometric width, line join, and line end attributes. The shape defined by the geometric line then is used for the clip path. The clip path can be a simple unmodified path.

`GpiSetClipPath` accepts two different path identifiers as input:

- 1
- 0 (default).

The default path identifier of 0, called `SCP_RESET`, resets the clip path to infinity, which displays the picture without clipping. If this value is selected, the current clip path definition is discarded instead of stored.

A path identifier of 1 is called `SCP_AND` and specifies that the clip path be redefined as the mathematical intersection of the stored clip path and the current path definition. The only method of specifying the clip path to the current path, after `GpiSetClipPath` has been issued, is to issue `GpiSetClipPath` twice: the first issuance with a path identifier of 0; the second, a path identifier of 1.

Figure 16-3 shows a triangle shape that has been defined within a path bracket and selected as the current clip path. The filled box shape is drawn subsequently, and, therefore, is clipped to the triangle.

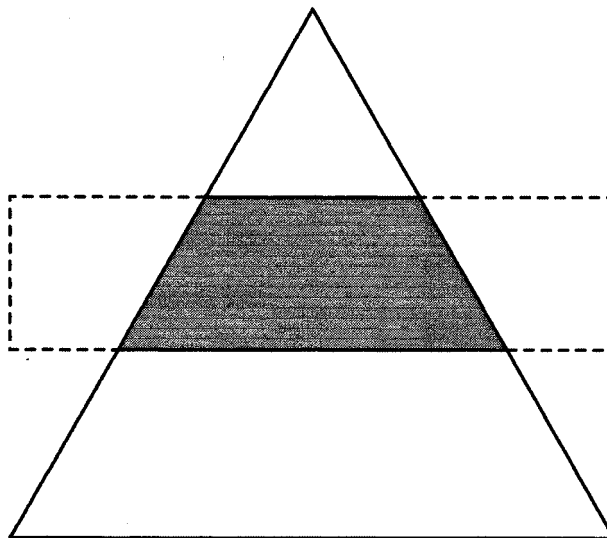


Figure 16-3. The Clipping Path. The broken lines show the area of the box that has been clipped.

Clip paths are most useful when you want to use an irregular clipping boundary, or when the clipping boundary itself is an integral part of the picture. Both are true of the clip path in Figure 16-3.

GpiSetClipPath also accepts 1 of 2 construction options as input:

- SCP_ALTERNATE (default)
- SCP_WINDING (must be selected if path has been modified).

Any drawing that is clipped to the current clip path must follow the alternate or the winding rules as to whether that portion of a picture is included in the clip area. The alternate and winding modes are described for paths in “Paths in Alternate Mode” and “Paths in Winding Mode.” Any point on the boundary of the path is considered within the path and is not clipped.

To end clipping to the current clip path, issue GpiSetClipPath with an identifier of 0. This function deselects the current clip path by setting it to infinity. In some circumstances, the current clip path is deselected automatically.

A path definition can be stored in a graphics segment; and, if that segment is retained, the path can be re-created as required when the segment is redrawn. Clip path definitions can be stored in a retained segment also and redrawn when required. In *draw-and-retain* mode, the initial path or clip path is created as the segment is constructed. If the current drawing mode is *retain*, however, the path or clip path is not created until the first time the segment is drawn.

The Viewing Window

The *viewing window* defines a rectangular clipping boundary in model space. It is defined with `GpiSetViewingLimits`. As input to this function, you supply the model coordinates of the lower-left and upper-right corners of the viewing window.

When a drawing primitive, such as a line, intersects a viewing window, any part of that line outside of the viewing window is clipped. Any point on the boundary of the viewing window is considered within the window, and is not clipped. By default, the viewing window performs no clipping. In this case, all graphics output in the model space is transformed. Figure 16-4 shows how the viewing window outlines a part of model space.

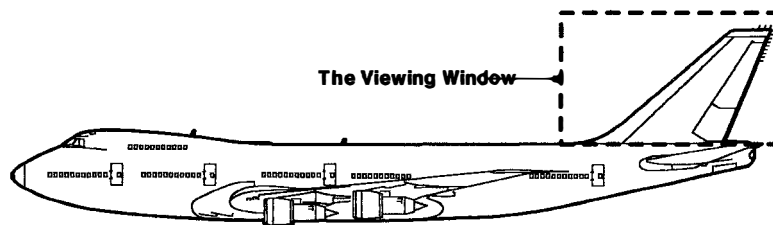


Figure 16-4. The Viewing Window. This example shows how the presentation page in Figure 17-15 is constructed. The viewing window outlines the tail of the aircraft, which is scaled and translated when drawn in presentation-page space. The rest of the aircraft is clipped away during the drawing process.

The Graphics Field

The graphics field defines a rectangular clipping area in the presentation page. It is defined in `GpiSetGraphicsField`.

Note: If this clipping area is to be used, it must be defined before any drawing begins.

Specify the size of the graphics field in presentation page coordinates as input to `GpiSetGraphicsField`.

Only the graphic output contained in this clipping boundary is visible when the presentation page is transformed to device space. By default, the graphics field is infinitely large and, therefore, performs no clipping. If you do specify a graphics field, however, any point on its boundary is considered within the graphics field and is not clipped. When a drawing primitive, such as a line, intersects a graphic field, any part of the line outside the graphics field is clipped. Figure 16-5 on page 16-6 shows how a graphics field could be defined for the presentation page in Figure 17-15.

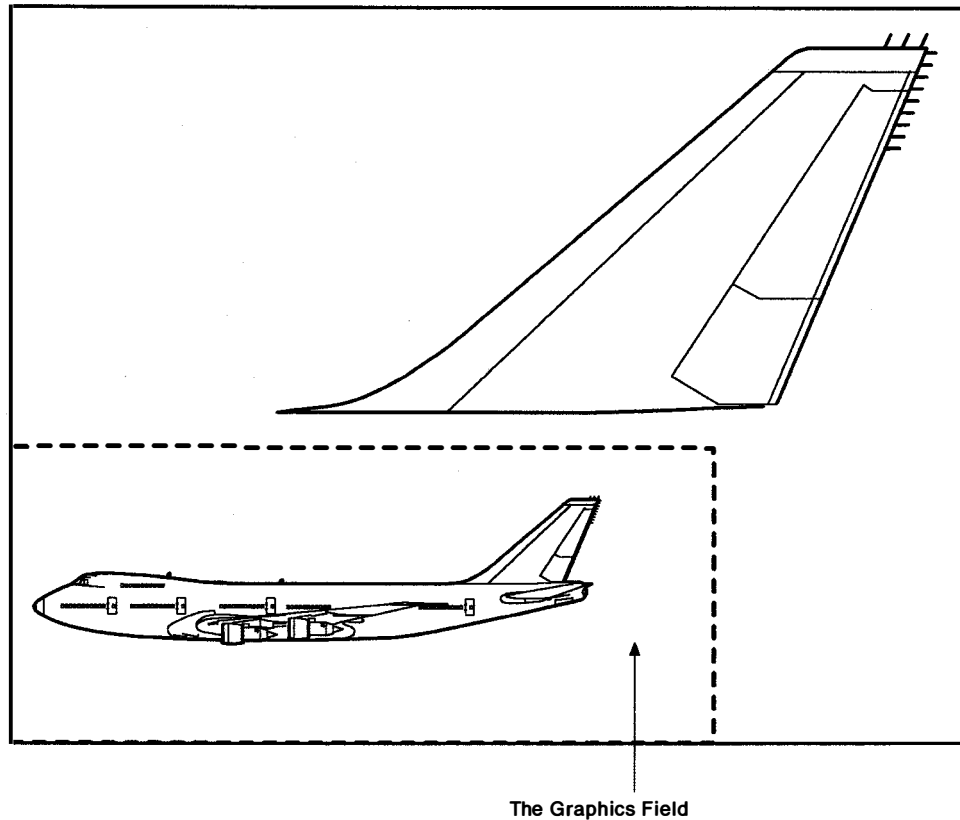


Figure 16-5. The Graphics Field. The broken line shows an arbitrary graphics field that is smaller than the presentation page. The aircraft tail, a separate object in the presentation page, is outside the graphics field and is clipped away as it is drawn.

The picture assembled in the graphics field is the picture that is displayed or printed. If you do not define a graphics field, the picture assembled in the presentation page is the picture that is displayed or printed. The presentation page is not a clipping boundary, and graphics in page coordinate space that are outside the presentation page boundary, therefore, might be visible.

The Current Clipping Region

Clipping regions are clipping areas defined (as regions) by one or more rectangles in device coordinates. Because they are defined in device coordinates, clipping regions do not suffer from the rounding errors associated with other types of clipping. Therefore, they are ideally suited to redraw part of the picture without boundary discontinuities, for example, after a BitBlt operation has been used to scroll a picture in a window.

Regions are not available automatically for clipping. To select an existing region as the current clipping region, use `GpiSetClipRegion`. By default, the clipping region is the same size as the drawing surface. Only one clipping region can exist in the presentation space at one time. To end clipping to the current clipping region, deselect it by issuing `GpiSetClipRegion` with a NULL region handle. A deselected clipping region retains the effects of any changes made to it while it was a clipping region and can be reselected.

You do not have to deselect the current clipping region before selecting another. Each selected clipping region automatically replaces the one before it. If a clipping region exists when you issue `GpiSetClipRegion`, the existing clipping region reverts to being a normal region, and its handle is returned.

Clip paths and clip regions share a common implementation, but clip regions are faster to create than clip paths. This might be a performance factor when designing your application for *repairing the screen*, or redrawing the picture in a client window after the display has changed. Figure 16-6 illustrates this use of regions.

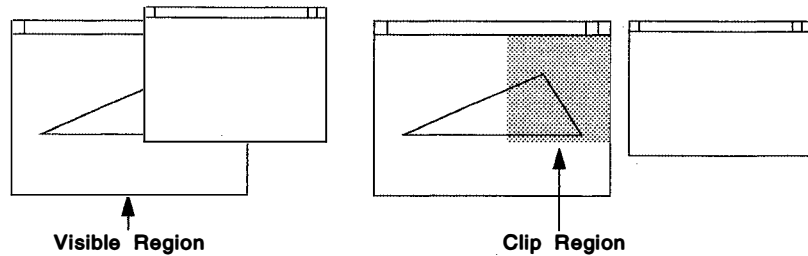


Figure 16-6. Screen Repairing

Note: `GpiSetClipRegion` does not cause graphics orders to be added to the current segment. Therefore, variations in the clip region must not be used to construct the picture. The clip region is intended to define a fixed clipping area for the entire picture.

When you select the current clipping region, none of the region-related GPI functions can be used for that region. The PM programming interface provides a series of functions that *mirror* the region-related functions. However, all of these functions work in world coordinates rather than device coordinates, and, therefore, are subject to current transformations.

Any of the following functions can be used to get information about or to redefine the current clipping region.

- **GpiQueryClipBox**

You can request the dimensions of the smallest rectangle that encloses all current clipping boundaries by issuing `GpiQueryClipBox`. The following boundaries are included in this calculation:

- Current clip path
- Current viewing window
- Current graphics field
- Current clipping region
- Visible region of the window.

- **GpiIntersectClipRectangle**

`GpiIntersectClipRectangle` redefines the current clipping region to the intersection of the existing clipping region with the rectangle whose dimensions you supply in this function. This has the same effect as `CRGN_AND` in `GpiCombineRegion`.

- **GpiExcludeClipRectangle**

You also can redefine the current clipping region using `GpiExcludeClipRectangle`. This function excludes a specified rectangle from the current region, and has the same effect as `CRGN_DIFF` on `GpiCombineRegion`.

- **GpiOffsetClipRegion**

The current clipping region can be moved from its current position using `GpiOffsetClipRegion`.

- **GpiPtVisible**

GpiPtVisible tells you whether a point, expressed in world coordinates, is visible on the screen. A point is visible if it is within all current clipping boundaries and is in the visible region of the window.

- **GpiRectVisible**

GpiRectVisible tells you whether any part or the whole of a rectangle, whose dimensions you supply in world coordinates, is visible on the screen. The rectangle is visible if it intersects both the visible region of the window and all current clipping boundaries.

How Clipping Is Implemented

The rules by which the PM programming interface implements clipping are as follows:

- Any primitive completely outside the clipping boundary is discarded.

When a primitive crosses a clipping boundary, any part outside the boundary is discarded.

- Any primitive completely within the clipping boundary is retained.

When a primitive crosses a clipping boundary, any part within the boundary is retained.

- When the clipping boundary is a clip path, viewing window, or graphics field, any point that falls on the boundary is considered within the clipping boundary.

When the clipping boundary is a clipping region, any point on the top or right boundaries of a rectangle is discarded; and any point on the bottom or left boundaries of a rectangle, that is not on the right or top boundaries also, is included in the region.

Redrawing Nondynamic Graphics

An interactive graphics application usually permits changes to the displayed picture, for example, an object can be moved or sized; and you can plan for this by defining particular segments as *dynamic*. Dynamic segments are described in Chapter 13, "Editing Retained Graphics and Graphics Segments."

If dynamic segments are inappropriate (when you are using nonretained graphics, for example), you can repair the picture using a clipping region, for example, a picture of a hexagon and a circle as shown in Figure 16-7.

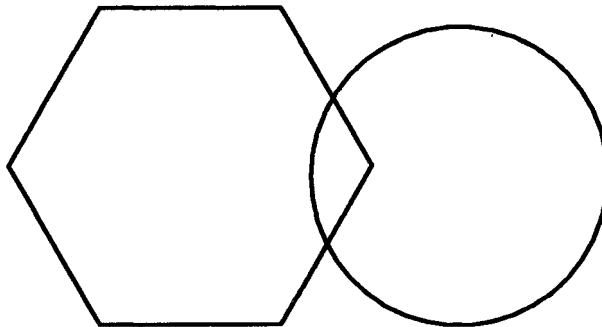


Figure 16-7. A Hexagon and Circle

If the circle is moved to another screen position by the use of an input device, you must repair its original location, and redraw it in its new location. Figure 16-8 shows this sequence of events. Following are the steps required to do this:

1. Determine the size of the smallest rectangle that contains the circle in its current position using a process called *boundary determination*.
2. Switch off the DCTL_DISPLAY flag of GpiSetDrawControl, apply a translation transformation to the circle, and redraw it in its new position.
3. Determine the size of the smallest rectangle that contains the circle in its new position using boundary determination.
4. Use GpiConvert to convert the model-space coordinates provided by the boundary-determination process to device-space coordinates.
5. Use the device-space coordinates of the two rectangles to create a region, and select it as the current clipping region.
6. Switch on the DCTL_DISPLAY flag of GpiSetDrawControl.
7. Issue GpiErase (or set the erase-before-draw control) to erase the current contents of the clipping region.
8. Redraw the picture with the circle in its new position. Any part of the picture within the clipping region is redrawn. That part of the hexagon that is outside the clipping region is unaffected by the change and does not have to be redrawn.

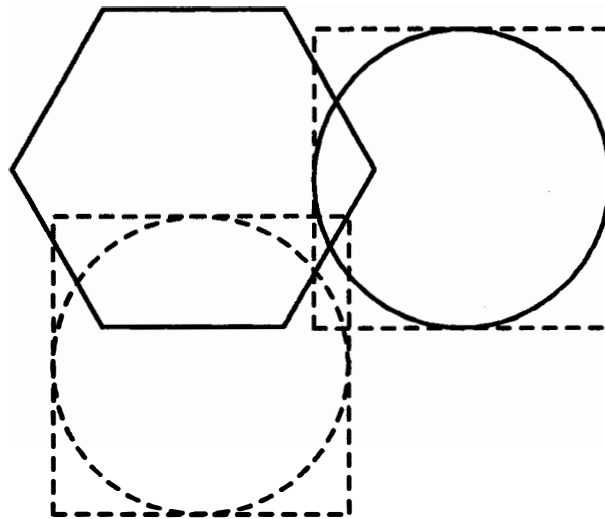


Figure 16-8. Defining a Clipping Region. The broken circle shows the position to which the circle is to be moved. The two bounding rectangles overlap, and produce a complex region. If the circle were to be moved much farther away from its start position, the region would comprise two disjoint rectangles.

About Boundary Determination

Boundary determination is an operation to compute the size of the smallest rectangle that encloses a graphics output in model space. One use of boundary determination is to enable you to repair only the affected parts of the screen, when a graphics object is moved, for example, or when a graphics object is changed some other way. Dynamic segments are not included in boundary-determination operations.

Boundary determination can be performed on both retained and nonretained graphics. In both instances, you request boundary data to be calculated by setting the *boundary data flag* (DCTL_BOUNDARY) in GpiSetDrawControl. If you do not set this flag (for example, if you do not want to collect boundary data unnecessarily) and later find that you need boundary data for a particular object, you can do the following:

1. Switch on the boundary-data flag, and switch off the display flag, using GpiSetDrawControl.
2. Redraw the object in its current location. Boundary data is collected, but the object is unaltered.

If you are drawing retained graphics, each drawing request (GpiDrawSegment, GpiDrawFrom, and GpiDrawChain) causes the boundary data resulting from the drawing to be made available. The application must request this data explicitly by issuing GpiQueryBoundaryData after each drawing request for which it wants to examine boundary data. Boundary data is returned to the application in model space coordinates. The boundary data is reset before each retained drawing operation, so there is no risk of accumulating data from separate operations.

If you are drawing nonretained graphics, boundary data is accumulated for each GpiPutData and for each individual primitive-drawing function. The application can request the accumulated boundary data at any time by issuing GpiQueryBoundaryData. Data continues to accumulate unless you issue the GpiResetBoundaryData; it is not reset automatically.

The boundary data returned to you is in the form of 4 model-space coordinates, which are the lowest (x,y) positions and the highest (x,y) positions of the bounding rectangle in model space, as illustrated in Figure 16-9.

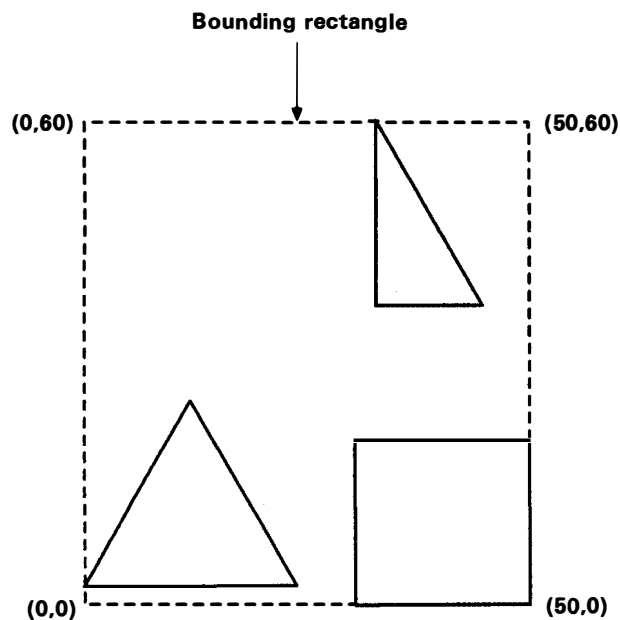


Figure 16-9. The Bounding Rectangle

Using Clipping and Boundary Determination

This section explains how to use clipping functions to:

- Create a clip path or clip region
- Exclude a rectangular area from a clip region
- Add a rectangular area to a clip region
- Set the clip region to the intersection of the current clip region and a specified rectangle
- Determine the size of the smallest rectangle that will surround the intersection of the current clipping areas completely.

Creating a Clip Path

A drawing and computer-aided-design (CAD) application may require the ability to clip to curved edges. If so, it must use a clip path to define a curved clipping area in world coordinates. Because clip paths (especially ones that clip to curved edges) require considerable memory and processing time, use them only when necessary; whenever possible, your application must use a clip region, graphics field, or viewing limit.

To create a clip path, do the following:

1. Determine the clip path's shape and size (in world coordinates).
2. Issue `GpiBeginPath` to begin defining the path.
3. Create the path.
4. Close the path definition with `GpiEndPath`.
5. Create a clip path from the path definition with `GpiSetClipPath`.

Figure 16-10 uses this procedure to create an elliptical clip path.

```
#define INCL_GPIPATHS
#include <os2.h>
void fncCLIP01(void){
    HPS    hps;                /* Presentation-space handle */
    POINTL ptl1;               /* Point structure           */
    FIXED  fxArc;              /* Multiplier for arc        */
    LONG   idPath;             /* Path identifier           */
    .
    .                          /* Load ptl1 with coordinates of clip path. */
    .
    idPath = 1;
    GpiBeginPath(hps, idPath);  /* Begins path                */
    GpiMove(hps, &ptl1);       /* Sets current position      */
    fxArc = MAKEFIXED(50, 0);  /* Sets arc multiplier        */
    GpiFullArc(hps, DRO_OUTLINE, fxArc); /* Defines ellipse          */
    GpiEndPath(hps);           /* Ends path                  */
    GpiSetClipPath(hps, idPath, SCP_ALTERNATE | SCP_AND);
} /* fncCLIP01 */
```

Figure 16-10. Creating a Clip Path

Creating a Clip Region

To create a clip region, first determine its size and shape (in device coordinates). Load the coordinates for the rectangles that define the clip region into an array of `RECTL` structures. Then create the region and set it to a clip region with `GpiCreateRegion` and `GpiSetClipRegion` respectively.

Figure 16-11 shows how to create a clip region.

```
#include <os2.h>
void fncCLIP02(void){
    HPS hps;
    HRGN hrgn;
    RECTL arcl[3];

    /* Load array of RECTL structures with coordinates. */

    hrgn = GpiCreateRegion(hps, sizeof(arcl) / sizeof(RECTL), arcl);
    GpiSetClipRegion(hps, hrgn, NULL);
} /* fncCLIP02 */
```

Figure 16-11. Creating a Clip Region

Excluding a Rectangular Area from a Clip Region

Some applications let you prepare output in multiple stages. For example, a word-processing application might permit you to prepare your text first and then add bit maps that enhance and support the text. These applications can use `GpiExcludeClipRectangle` to exclude an area from a clip region, preventing the user from deleting output that already exists.

To exclude an area from a clip region, first determine the dimensions (in device coordinates) of the smallest rectangle that completely surrounds the area to exclude from the clip region. Then, issue `GpiExcludeClipRectangle`, including, as input, the dimensions of the area to exclude. Figure 16-12 illustrates these steps.

```
#include <os2.h>
void fncCLIP03(void){
    HPS hps;
    RECTL rcl;

    /* Set rectangle coordinates here. */

    GpiExcludeClipRectangle(hps, &rcl);
} /* fncCLIP03 */
```

Figure 16-12. Excluding a Rectangular Area from a Clip Region

Adding a Rectangular Area to a Clip Region

Some applications might need to increase the size of a clip region. For example, a user might request that a desktop-publishing application extend a column of text on a page.

To add a rectangular area to a clip region, follow these steps:

1. Determine the dimensions (in device coordinates) of the rectangular area to add to the current clip region.
2. Release the current clip region using `GpiSetClipRegion`. `GpiCombineRegion` cannot combine regions if either of the regions is a clip region.
3. Issue `GpiCreateRegion` and pass it the dimensions of the rectangle that you defined in Step 1. This creates a second region that you can combine with the first.
4. Reissue `GpiCreateRegion` and create a third region that will be the final destination region.
5. Issue `GpiCombineRegion` to create a region that combines the original region and the region you created in Step 2. It is not essential to create a third region because `GpiCombineRegion` can use one of the 2 source regions being combined for a target region.
6. Issue `GpiSetClipRegion` and pass it the handle returned by `GpiCombineRegion`.

Figure 16-13 illustrates these steps.

```
#define INCL_GPIREGIONS
#include <os2.h>
void fncCLIP04(void){
    HPS    hps;
    RECTL  rc11, rc12, rc13;
    HRGN   hrgn1, hrgn2, hrgn3;

    hrgn1 = GpiCreateRegion(hps, sizeof(rc11) / sizeof(RECTL), &rc11);
    GpiSetClipRegion(hps, hrgn1, NULL);          /* Creates first clipping region */

    /* Compute coordinates of second region here. */

    GpiSetClipRegion(hps, NULLHANDLE, NULL); /* Releases first clipping region*/
    hrgn2 = GpiCreateRegion(hps, sizeof(rc12) / sizeof(RECTL), &rc12);
    hrgn3 = GpiCreateRegion(hps, sizeof(rc13) / sizeof(RECTL), &rc13);
    GpiCombineRegion(hps, hrgn3, hrgn1, hrgn2, CRGN_OR);
    GpiSetClipRegion(hps, hrgn3, NULL);          /* Creates second clipping region*/
} /* fncCLIP04 */
```

Figure 16-13. Adding a Rectangular Area to a Clip Region

This operation also could be performed with `GpiIntersectClipRectangle`.

Setting the Clip Region to a Region Intersection

Your application might require the ability to set the clip region to the intersection of the current clip region and another region. Do this with `GpiIntersectClipRectangle`, as shown in Figure 16-14.

```
#include <os2.h>
void fncCLIP05(void){
    RECTL rcl;
    HPS hps;

    /* Load rcl with coordinates of rectangle to intersect. */

    GpiIntersectClipRectangle(hps, &rcl);
} /* fncCLIP05 */
```

Figure 16-14. Setting the Clip Region to a Region Intersection

Determining the Size of a Clipping Area

If an application is able to specify a clip path in world space, a viewing limit in model space, a graphics field in page space, and a clip region in device space, it might be necessary for your application to determine the size of the clipping area formed by the intersection of the four. `GpiQueryClipBox` returns the dimensions (in world coordinates) of the smallest rectangle that completely surrounds the intersection of all the defined clipping areas, including the visible region.

Figure 16-15 shows how to use `GpiQueryClipBox` to fill a `RECTL` structure with the desired coordinates.

```
#include <os2.h>
void fncCLIP06(void){
    HPS hps;
    RECTL rclClip;
    GpiQueryClipBox(hps, &rclClip);
} /* fncCLIP06 */
```

Figure 16-15. Determining the Size of a Clipping Area

Summary

Table 16-2 summarizes the functions used to perform clipping.

<i>Table 16-2. Clipping Functions</i>	
Function Name	Description
GpiExcludeClipRectangle	Excludes a specified rectangle from the current clipping region.
GpiIntersectClipRectangle	Combines a rectangle with the current clipping region to form a new clipping region.
GpiOffsetClipRegion	Moves the clipping region by a specified amount.
GpiPtVisible	Determines whether a point, given in world coordinates, is within all current clipping boundaries.
GpiQueryClipBox	Determines the smallest rectangle that encloses all current clipping boundaries.
GpiQueryClipRegion	Determines the handle of the currently selected clip region.
GpiQueryGraphicsField	Determines the location, in page coordinates, of the lower-left and upper-right corners of the current graphics field rectangle.
GpiQueryViewingLimits	Determines the location, in model coordinates, of the lower-left and upper-right corners of the current viewing limits rectangle.
GpiRectVisible	Determines whether any part or the whole of a rectangle, given in world coordinates, is within all current clipping boundaries.
GpiSetClipPath	Defines the current path as a clip path.
GpiSetClipRegion	Defines the current region as a clip region.
GpiSetGraphicsField	Defines the rectangle in presentation-page space as the graphics field clipping boundary.
GpiSetViewingLimits	Defines the rectangle in model space as the viewing limits clipping boundary.

Table 16-3 summarizes the data structure used by the functions that perform clipping.

<i>Table 16-3. Clipping Structure</i>	
Structure Name	Description
RECTL	A structure specifying the values of two (x,y) coordinate pairs, defining the bottom-left and top-right coordinates of a rectangle.

Chapter 17. Coordinate Spaces and Transformations

A *transformation* is an operation performed on a graphic object that changes the object in one of four ways: translation, rotation, scaling and shearing. Transformations enable an application to control the location, orientation, size, and shape of graphics output on any output device.

The transformation of graphic output can be conceptually divided into a series of distinct transformations applied from 1 logical stopping point to another. Coordinate spaces are used as a method of conceptualizing these logical stopping points. The coordinate spaces are concepts used to explain and manipulate the transformation process.

A graphic primitive in an intermediate coordinate space cannot be displayed on an output device and a transformation cannot be interrupted at 1 of the distinct stages. The entire series of transformation steps, is applied all at once.

This chapter describes the transformation design process. The following topics are related to the information in this chapter:

- Presentation spaces
- Device contexts
- Segments and retained graphics
- Clipping.

About Coordinate Spaces

Most of the GPI functions draw their output in a conceptual area called the *world coordinate space*. If you picture the presentation space as a blank canvas on which to draw, the world coordinate space is a Cartesian grid that provides a reference of scale for what is being drawn.

The components of a picture defined in a world coordinate space are often defined to a scale convenient only to that component. Applications also can define each component, or *subpicture*, starting at the origin (0,0). This enables applications to define the scale of a subpicture and the location of the subpicture separately. The ability to define all subpictures at the origin means there is not always 1-to-1 correspondence between a presentation space and a world coordinate space. Frequently a separate world coordinate space exists for every subpicture.

After subpictures are defined in the world coordinate space, they undergo a process called transformation then appear on an output device. If an application has not specifically applied a transformation to its subpictures, by default the PM programming interface applies the *identity transformation*, which, in effect, makes no changes to the subpicture.

All graphics systems, not just the PM programming interface, use at least 1 coordinate space to generate output. The simplest graphics systems use a single coordinate space, whose points are the pels on the display. These are sometimes called single-space systems.

The PM programming interface creates multiple coordinate spaces and uses transformations to move subpictures between the coordinate spaces. The PM programming interface assembles the application's graphic output in a process that can use up to 5 coordinate spaces, known collectively as the *viewing pipeline*. The PM programming interface coordinate spaces are:

- World Coordinate Space
- Model Space
- Page Space
- Device Space
- Media Space.

For those who are more familiar with a graphics system that uses a single-coordinate space, Table 17-1 lists the differences to expect when using PM's multi-coordinate space.

<i>Table 17-1. Single-Coordinate vs. Multi-Coordinate Space Systems</i>	
Single-Coordinate Space Systems	PM Multi-Coordinate System
Distances and locations specified in pels.	Seven units of measurement, including pels, are available.
Because the shape, or aspect ratio, and size of pels can vary on different devices, the graphic primitives on 1 device might look different on another.	The output is device independent.
Coordinates entered must already be device coordinates.	Coordinates are made device compatible through the PM programming interface.

Additionally, a single-space system requires that applications keep track of:

- The scaling between the subpictures. Often this is accomplished by forcing the definition of objects to the scale of the picture, instead of to the scale of the subpicture.
- The scaling between the creation space and output space. Often this is accomplished by forcing the definition of objects to the scale of the paper or window, instead of to the scale of the subpicture.
- All aspects of other transformations.

World Coordinate Space

The world coordinate space is where most drawing coordinates are specified. Each primitive in a world coordinate space is defined using the coordinate positions in the primitive's definition. For example, if an application draws a line from (8,4) to (20,75), the coordinate values (8,4) and (20,75) are world coordinates.

The world coordinate space is specified when the presentation space is created with GpiCreatePS. The PS_FORMAT option supports 2 sizes of world coordinate spaces: short and long. If the size is short (GPIF_SHORT), the world coordinate space is a rectangular Cartesian space with maximum coordinates of (32767, 32767) and minimum coordinates of (-32768, -32768). If the world coordinate space is short, it is the application's responsibility to ensure that the coordinates defining the subpicture fall within this range.

The second size, long (GPIF_LONG), is the default for the world coordinate space. Most GPI functions that direct their output to the world coordinate space, for example GpiLine, accept 32-bit integers as parameters. A 32-bit integer is

equivalent to a rectangular Cartesian space with maximum coordinates of (134217727, 134217727) and minimum coordinates of (-134217728, -134217728).

World coordinates do not have to be within the range of the presentation page. A graphic image that requires a high degree of detail and precision might use most or all of the available coordinate range. A transformation is used to scale the graphic image down to an appropriate size. The actual presentation page size is unimportant in most cases. It is used with the page viewport to redefine the device transform if `GpiSetPageViewport` is called.

An application can use a *clip path* clipping area to define the part of the world space to place in the next coordinate space (the model space). A clip path is the only clipping region that can be nonrectangular. Its edges can include arcs, curves, and straight lines. The coordinates that define the dimensions and shape of a clip path are always world coordinates. (Clipping is further described in Chapter 16, "Clipping and Boundary Determination.")

In a world-coordinate space, there can be several graphic primitives. If, however, an application uses the `DM_RETAIN` drawing mode to store output in graphic segments, the operating system assigns a new world space to each segment. There is also a world space for the drawings outside of segments.

Model Space

Model space is the conceptual area where the separate components of a picture, defined in world space, are brought together. To assemble 1 or more primitives from world spaces to model space the application specifies the transformations to occur, then the PM programming interface applies them to each of the components. Model transformations convert world coordinates to model-space coordinates. For example, an octagon and the word STOP, defined individually in separate world-coordinate spaces, can be assembled into a stop sign in model space.

Graphics applications can have more than 1 model space. If there is more than 1 model space, the picture components are assembled in page space. If an application has each model space in a different segment, the model transforms are reset for each segment.

An application can use a *viewing limit* clipping area to define the part of a model space to place in the next coordinate space (the page space). A viewing limit is always rectangular, and the coordinates that define its location and dimensions are always model coordinates.

Page Space

The page space is where a complete picture is assembled for viewing on a display screen, or for printing or plotting on a piece of paper. Page coordinate units can be increments of an inch, a meter, pels, or some arbitrary value. An application uses `GpiCreatePS` to specify the units used for page coordinates.

If the application uses retained-drawing mode, the picture in page space contains parts of models (or pictures) from each of the model spaces that have not been clipped. The picture also contains any additional unclipped graphics-primitive output that the application generated in nonretained-drawing mode. If the application uses nonretained-drawing mode, the picture in page space contains all the graphics-primitive output that has not been clipped.

In the page space, an application can use a rectangular clipping area called a *graphics field* to define the part of the page space to place in the next coordinate space (the device space). The coordinates that define the location and dimensions of the graphics field are always page coordinates.

The page space contains the presentation page whose size and units are defined when creating the presentation space with `GpiCreatePS`. The presentation page is a rectangle in page space.

Device Space

The device space is the coordinate space in which a picture is drawn before it appears in a display screen window or on the printer or plotter.

Device space is defined in device-specific units. Depending on the page unit used, device-coordinate units can be pels, increments of an inch, increments of a meter, or arbitrary. For example, if the page unit is pels, the device-coordinate unit is pels; if the page units are arbitrary, the device-coordinate units are arbitrary.

Media Space

The media space is used only with windows. When an application draws a primitive at a specified location in a window, it is not actually drawing in the device space, as the device is the entire terminal screen.

Drawing in a window involves a shifting transformation which moves a drawing from the given (unitless) position, to position in the specified window.

About Transformations

The coordinate spaces are connected by different transformation functions. In reality, the input coordinates of the world coordinate space are transformed directly into device coordinates in a single operation. The intermediate coordinate spaces exist only to provide a useful model to assist in the understanding of how to define the different transformation matrixes.

Transformation of graphic primitives occur when a transformation matrix is applied to those primitives. The individual transformations introduced here, do not actually transform the primitives, but rather define portions of the transformation matrix. After all portions of the matrix are identified, the actual operation is performed in a single step.

The transformation functions manipulate graphics primitives as they move from one coordinate space to the next. Transformation functions usually begin with the letters *Gpi*. Most of the function names have a *transformation type* that identifies the entities on which it operates. For example, a model transformation is the transformation type that transforms graphics primitives between a world coordinate space and a model space. There is a 1-to-1 correspondence between these transformation types and the actual functions. The transformation matrix data structure is called *MATRIXLF*.

Figure 17-1 on page 17-5 lists the sequence of coordinate spaces and the transformation types between the coordinate spaces. Internally, all transformations are combined, and the resulting values are held in the same format as the individual components. By default, there is no viewing window and no graphics field. The application can use the default page viewport. Clipping regions for each coordinate space are also shown.

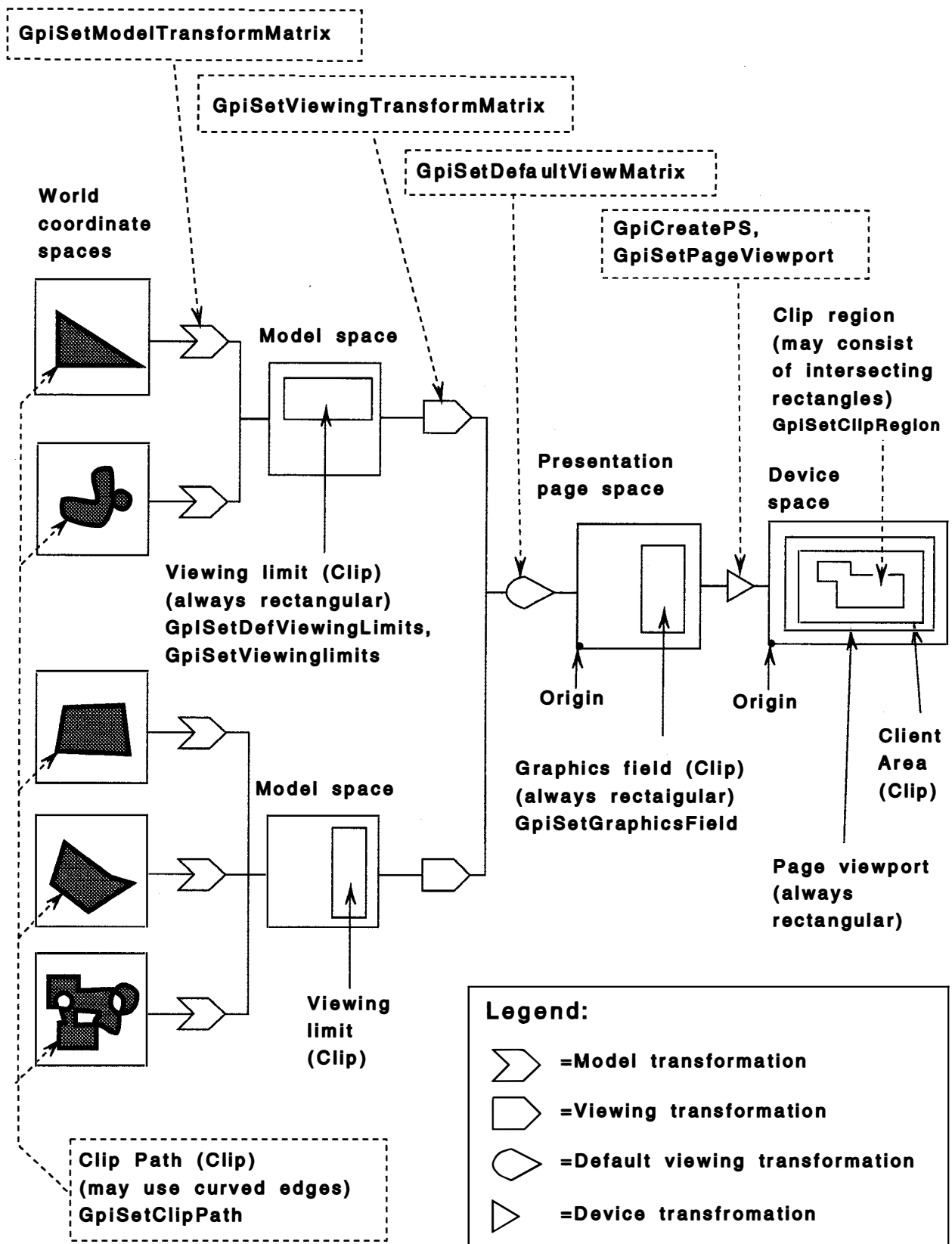


Figure 17-1. Viewing Pipeline

Identity Transformation

The *identity transformation* is the default transformation between all coordinate spaces. The identity transformation, makes no change to the original coordinates of an object. This transformation is also referred to as the *unity transformation*, because of the mathematical matrix used to make this transformation.

The identity matrix looks like this:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(Transformations are accomplished with matrix multiplication; and 1, not 0, is the multiplication identity.)

If a transformation is not explicitly specified, the identity transformation is used to create that portion of the transformation matrix. Most transformations applied to a primitive can be in addition to, or instead of, the current transformation.

Moving through Coordinate Spaces Using the Identity Transformation

The effects of transformation can be minimized by defaulting to the identity transformation. This means that no deliberate action is requested by the application.

Graphic primitives are drawn in their own coordinate scale in world coordinate space. The primitives are then combined in model space. Neither the world coordinate space or the model space have “real world” units associated with them. This means that if an application draws a line on the x-axis from -300000 to $+300000$, the line exists as a line 600000 Cartesian units long in both world coordinate and model space.

Page space is the first of the coordinate spaces to be associated with units such as millimeters or inches. These units are set with the GpiCreatePS option PS_UNITS. If the application specifies the option PS_UNITS with the value PU_ARBITRARY, the page space remains unitless; units are applied when the output is drawn in device space.

As an example, assume that the units PU_LOMETRIC, (0.1 mm) have been chosen. Again, assuming the identity transformation between model and page space, the primitive appears in the page space as a line that extends

$300000 * 0.1\text{mm}$ to the right of the origin
 $300000 * 0.1\text{mm}$ to the left of the origin

A presentation page, a rectangle in page space, has its size defined when a presentation space is created with GpiCreatePS. The presentation page format is defined with the GpiCreatePS option, PS_FORMAT. The 3 choices for that option are:

- GPIF_DEFAULT—32-bit integers or 2GB
- GPIF_LONG—32-bit integers
- GPIF_SHORT—16-bit integers or 32KB.

PS_FORMAT effects the number of units permitted along the coordinate axes of the presentation page. If the choice was GPIF_LONG or GPIF_DEFAULT, and, for example, PS_UNITS remains PU_LOMETRIC, the presentation page axes could

range from 0 to 134217727 0.1mm. If the choice was GPIF_SHORT, the coordinate axes could range only from 0 to 32767 0.1mm. These limits are the maximum number of units, presentation pages are usually much smaller.

The presentation page does not effect the primitive in presentation space, it determines what parts of the primitive are visible. With no transformation to scale down the primitive from the model to the page space, the presentation page acts like a sheet of cardboard with a rectangular hole. Only the part of the page space behind the "hole" is visible in page space. The view of everything else is blocked by the "cardboard." Only what is visible in the page space is drawn to the next coordinate space.

Continuing with the example of the 600000 units line above, if GPIF_LONG (or GPIF_DEFAULT) is selected, the 600000 unit line could easily be viewed in the presentation page. If GPIF_SHORT is selected, the maximum presentation page size is 65534 * 0.1 mm units long. In the case, much of the 600000 unit line would not be drawn into the next coordinate space.

The device space is not affected by the PS_FORMAT option. Whether GPIF_LONG, or GPIF_SHORT, the device space is a rectangle with maximum coordinates of (32767, 32767) and minimum coordinates of (-32768, -32768). Note that not all of this rectangle is visible, since real devices aren't that big. The device space relates to the physical device.

The transformation between the page and device space is handled automatically by the PM programming interface. The presentation page is mapped into a rectangle in the device space called the page viewport. This transformation is based on the parameters and options of the GpiCreatePS function.

Confusion can arise because the effects of the presentation page and the page viewport appear to be a clipping of the primitives in page and device space. While the action taken is identical, the term clipping is reserved for the activity deliberately designated by the application with clipping functions.

Applying Transformations Other Than the Identity Transformation

To better understand the workings of transformations other than the identity transformation, an example of the picture assembly process is illustrated in Figure 17-2, and a detailed explanation follows.

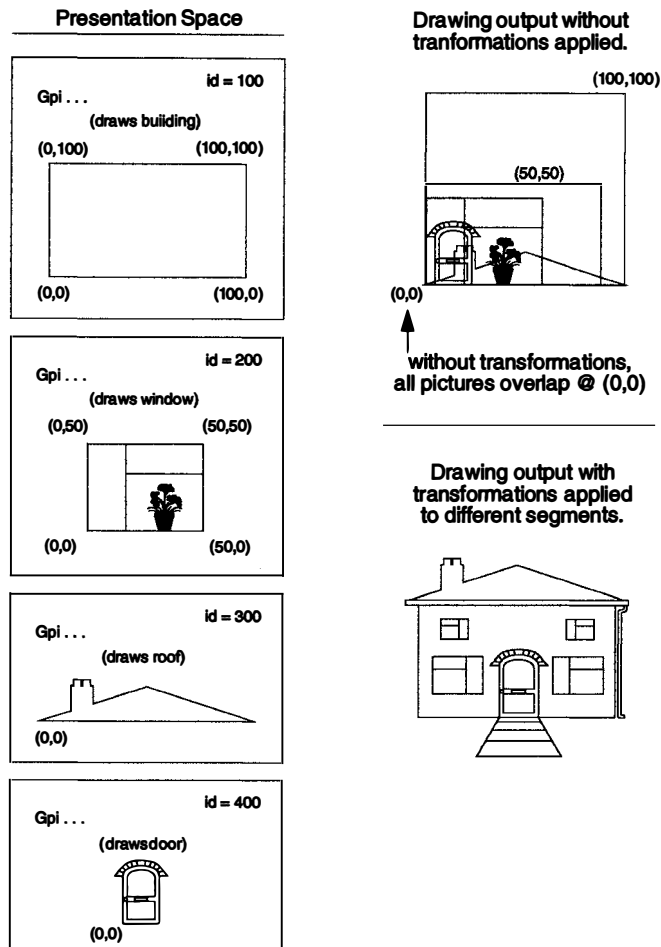


Figure 17-2. Picture Assembly Process

In the world space of Figure 17-2 four segments are drawn, each containing a different subpicture. The units of the subpictures can be different (for example, the building might be measured in feet, while the window might be measured in inches), because each subpicture is converted to the scale of the completed picture when it is transformed into the completed picture. All of the subpictures in Figure 17-2 are defined at “real world” scales in their own (Cartesian) world coordinate space.

The difference between applying and not applying transformations can be seen in the model space in Figure 17-2. Without transformations, the subpictures would be drawn at exactly the coordinates given in the world space and thus all four subpictures would overlap. With transformations, the subpictures can be scaled and translated to the scale of the final picture. The window subpicture, Segment id=200 has been scaled, rotated, and translated.

The model space contains the model of what the application is trying to draw. There can be more than 1 model space for very complex drawings, but this is not a recommended style of programming.

The building also resides in the page space, and is prepared in the device space for printing on an 8½ by 11 sheet of paper. This is shown in Figure 17-3 on page 17-9. Most often the page space to device space transformation (the device transformation) is the identity transformation.

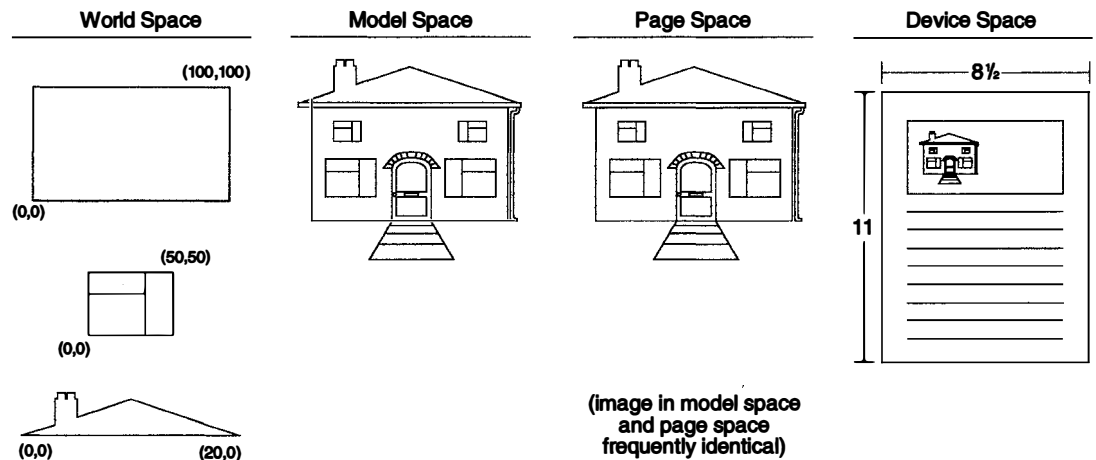


Figure 17-3. Different Spaces

The transformation process in Figure 17-3 is acting on segments. To draw the composite picture in model space, create a segment chain that plays all 4 segments associated with the building. The purpose of the world to model space transformation is to transform the graphic segments into a composite picture.

Having built the composite picture, the next step is to map the composite into the page space. Usually, the model-to-page space transformation is the identity transformation. The different coordinate spaces are shown in Figure 17-3

The user might want an exploded diagram as well as the composite picture. This is illustrated in Figure 17-4 on page 17-10. To create the exploded diagram, the application must draw the segment chain again, with a slight translation down, and scaled up (enlarged) with a very large scaling factor.

Again, while it is possible to create several complex images in different model spaces that must be assembled in the page space, the most frequent use of the page space is to prepare a view of the model-from-model space with the first application of "real world" units such as inches or millimeters. As shown in the figures above, both world and model space share unitless Cartesian coordinates.

The usual purpose of the page space is to show multiple views of the image residing in the model space, as shown in Figure 17-4.

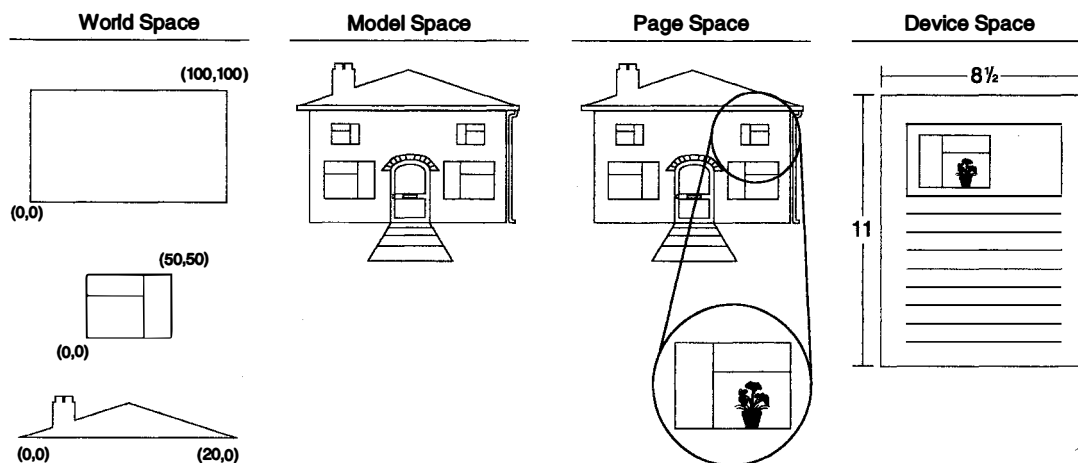


Figure 17-4. Different Spaces with an Exploded View

The enlarged image on the right side of the page space is drawn after the image on the left. It therefore has a higher priority, and therefore would be drawn on top of the left side image. To obtain a multi-viewed page space image, the application defines a clipping region, and applies this region to the second (enlarged) playback of the segment chain. The clipping region permits the enlarged image to appear only on the right side of the page space.

The same building also resides in this model space; but, it resides in the page space, along with an exploded view of the uppermost right side window. The view is enlarged to such a magnitude that the details of the window are once again evident. If the images in world coordinate space are not able to be drawn in the amount of detail now supported by the PM programming interface, then drawing details, such as an exploded view, would reveal the barrenness of the image.

Subpictures can be drawn in world coordinate space without being drawn inside a graphics segment.

Combining Transformations Between a Coordinate Space Pair

As illustrated in Figure 17-1, there are more than 1 transformation between the world coordinate and model spaces, and between the model and page spaces. Depending on the desired output, the transformations can be combined in different ways.

Between the world coordinate and model space, there are 5 ways of combining the 3 transformations.

<i>Table 17-2. World Coordinate to Model Space Transformation Combinations</i>		
Transformation Function	Effect	Transformation Type Sequence
GpiCallSegMatrix	ADD	M I S
	PREEMPT	I M S
	REPLACE	I S
GpiPlaySegment (Call Segment)	Drawing outside of an instance drawing primitive but inside a segment.	M S
No specific function	Drawing outside of segments altogether.	M
Abbreviations: I - Instance Transformation, M - Model Transformation, S - Segment Transformation		

Between the model and page space pair are 2 transformations and they also can be combined in different ways.

<i>Table 17-3. Model to Page Space Transformation Combinations</i>		
Transformation Function	Effect	Transformation Type Sequence
GpiSetPageViewport	Drawing inside of and outside of segments.	V D
GpiSetDefaultViewMatrix	Drawing outside of segments.	D
Abbreviations: V - Viewing Transformation, D - Default Viewing Transformation		

Transformation Mathematics

The transformation of a picture can be represented in general terms by 2 linear equations that define how the (x,y) coordinates of each point in the picture are changed.

The general form of these equations is: $x' = Ax + Cy + E$ $y' = Bx + Dy + F$	where (x, y) defines the original point (x', y') is the transformed point A, B, C, D, E, and F are constants.
--	---

The transformations that result from these equations depend on the values of the constants, which in turn vary according to the type of transformation being applied.

The operating system handles transformations by using matrix mathematics. In the matrix mathematics,

- Translation is an addition operation
- Scaling, reflecting, rotation, and shear are multiplication operations.

If all the transformations were multiplication operations, different types of transformation could be applied with a single transformation operation. Therefore, to facilitate the combining of calls, translation in the OS/2 operating system is performed as a multiplication operation, rather than an addition operation.

This requires that the vector representing a point, $[x \ y]$ be extended by a third component, w : $[x \ y \ w]$.

This enables all the transformations to be handled in a uniform manner.

This is called a *homogeneous coordinate system*. The value, w , is a multiplier, so that the point represented is: (wx, wy) .

Note: The PM programming interface does not support a 3-dimensional presentation space. The 3-dimensional matrix is created to effect matrix multiplication.

In this notation, the point (x, y) is represented as $[x \ y \ 1]$. To be able to operate on such three-element vectors, and to combine translation with the multiplication operations, the 2-by-2 matrix has to be extended to a 3-by-3 matrix, with the third column being:

0
0
1

The linear equations: $x' = Ax + Cy + E$ $y' = Bx + Dy + F$	become the matrix equations: $[x' \ y' \ 1] = [x \ y \ 1] * \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ E & F & 1 \end{bmatrix}$
--	---

In standard matrix notation, you have:

$$[x' \ y' \ 1] = [x \ y \ 1] * \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

Model for Building the Transformation Matrix

The transformation matrix shown above can be set equal to an entity as complex as:

$$M * I * S * V * D$$

where $*$ symbolizes matrix multiplication. As mentioned earlier, all the transformation types might not have a distinct value at all times. One, or all, could simply be the identity transformation.

These intermediate space and step-wise transformations are not actually how the operating system performs transformations. The reason for this relates to the way the PM programming interface stores transformation information. The transformation matrix data structure MATRIXLF, undergoes continuous modification during the drawing process, and it can be influenced by more than 1 transformation function from the application responding, for example, to what a user wants the drawing to look like.

For example, after creating the model of the building (Figure 17-2), from drawing primitives, the user might want to see 2 identical buildings side-by-side. The

application has already applied a series of transformations to the 2 segments in world coordinate space to create the model of the building. If the application, rather than the PM programming interface, had to perform the entire transformation over again from start to finish, it would have to keep track of much more detail. The PM programming interface enables applications to process a smaller portion of the MATRIXLF structure, in this example the “V” or “V and D” component, to enable the user to see 2 buildings side-by-side.

MATRIXLF, the Transformation Matrix

For all the transformation functions, the 3-by-3 transformation matrix is specified as a one-dimensional array in the following order:

$(M_{11}, M_{12}, 0, M_{21}, M_{22}, 0, M_{31}, M_{32}, 1)$

This one-dimensional array corresponds to the data structure, MATRIXLF. The elements in the data structure correspond to the matrix multiplication constants, as shown in Table 17-4.

<i>Table 17-4. Transformation Factors in the MATRIXLF Data Structure.</i>	
Transformation Type	Matrix Values
Scaling, Reflection	M_{11} and M_{22} .
Rotation	$M_{11}, M_{12}, M_{21}, M_{22}$
Translation	M_{31} and M_{32}

All nine elements do not have to be specified for every transformation function. However, those specified are interpreted as the first n of the nine. If the third, sixth, and ninth elements are specified, they must be 0, 0, and 1 respectively.

MATRIXLF Structure

Of the nine fields in MATRIXLF, four are special 32-bit FIXED variables. The remaining 5 are 32-bit long integer variables.

The scaling, reflecting, and rotation constants, $M_{11}, M_{12}, M_{21}, M_{22}$, are the 32-bit, signed, FIXED variables, with an implied binary point between the second and third bytes. The translation constants, M_{31} and M_{32} , and the 3 third-column variables, are 32-bit long integers.

A FIXED variable is a binary representation of a floating-point number. A FIXED variable has 2 parts:

- The high-order 16-bits which contain a signed integer in the range -32768 through 32767
- The low-order 16-bits which contain the numerator of a fraction in the range 0 through 65535. The denominator for this fraction is 65536.

For example, to store the cosine of 60° (0.5) in a FIXED variable, an application would multiply 65536 by 0.5. The result, 32768, would be the value to assign to a field in the MATRIXLF structure.

To store a scaling factor of 3 in a FIXED variable, the application would multiply 65536 by 3. Again, the result, 196608, would be the value to assign to a field in the MATRIXLF structure.

MAKEFIXED Macro

The MAKEFIXED macro provides a quick and convenient method for setting the value of FIXED variables. This macro requires 2 arguments: the first is the integer part of the FIXED value, and the second is the fraction part of the FIXED value. In the following example, MAKEFIXED is used to assign the FIXED value equivalent of $1 \frac{1}{8}$ to the matrix component M_{11} .

```
matlf.fxM11 = MAKEFIXED(1, 8192)
```

Figure 17-5. Determining a FIXED Equivalent

The first argument, 1, is the integer part of the FIXED value. The second argument, 8192, is the result of multiplying 65536 by $\frac{1}{8}$.

If it is necessary for an application to scale or rotate an object, the application can avoid most of these mathematics by using the helper functions, GpiRotate and GpiScale. GpiRotate accepts a rotation in degrees and converts this value into the appropriate fields in the MATRIXLF structure. Similarly, GpiScale accepts a scaling factor and fills in the MATRIXLF structure with the appropriate values.

About Transformation Operations

The available transformations are listed in Table 17-5.

Table 17-5. Transformations	
Operation	Result
Scaling	Shrinks or enlarges the object
Reflection	Creates a mirror image of an object with respect to the x- or y-axis
Rotation	Rotates the object
Translation	Shifts the object with respect to the origin of the coordinate system
Shear	Rotates either all the vertical or all the horizontal lines in an object

These basic operations can be combined.

The PM programming interface provides special functions to perform scaling, rotation, and translation and also enables applications to specify the transformation matrix directly. Applications can specify values for more than 1 type of transformation on a single transformation call.

Transformations are used to manipulate graphic objects as they are being moved from 1 coordinate space to another. These operations are performed by functions called *transformation functions*. There are also functions that help perform the transformations called *helper functions*.

The scaling, reflection, rotation, translation, and shear transformations are best demonstrated by applying them to a picture. Figure 17-6 on page 17-15 shows the image of a flag before any transformations have been applied. The flag is defined by 5 points. Their (x,y) coordinates are (0,0), (0,4), (0,6), (2,6), and (2,4).

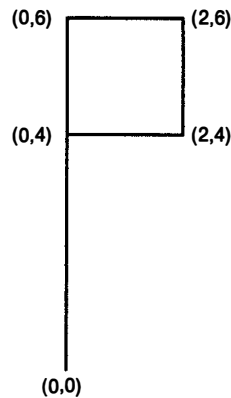


Figure 17-6. Flag Before Transformation

In the next several sections, where the transformations are described in detail, the effect of the transformation on the flag is illustrated.

Scaling and Reflection Transformations

Applications can scale an object by using `GpiScale` or by modifying the `MATRIXLF` structure directly. A scaling transformation reduces or increases the size of a graphics object. A reflecting transformation creates a mirror image of an object with respect to the x- or y-axis.

A scaling factor of:

- greater than 1 causes an increase in size
- greater than 0 but less than 1 causes a reduction in size,
- less than 0 causes a reflection about that axis. That is, a negative x scaling factor causes reflection in the x direction.

Note: If an application specifies a scaling factor of greater than 1, the graphics presentation space must be defined with the coordinate format `GPIF_LONG`. This is because 32-bit matrix elements are required to store these values in retained segment and metafile orders.

The equations to scale by factors S_x and S_y are obtained from the general equations (with $M_{11} = S_x$ and $M_{22} = S_y$) and can be written:

$$x' = xS_x$$

$$y' = yS_y$$

A scaling transformation reduces or increases all the coordinates of an object by the scaling factor. Any object not aligned on the x- and y-axes is therefore moved nearer to the origin by a reduction in size, and away from the origin by an increase in size. For example, if an application applies a scaling factor of 0.5 to a simple box with its corners at (4,4), (10,4), (10,10), and (4,10), the four corners moves to (2,2), (5,2), (5,5), and (2,5).

To scale an object about a point without causing it to move, the following sequence of transformations is required:

1. Translate the scaling point of the object to the origin.
2. Scale the object at the origin.
3. Translate the scaling point of the object back to its original position.

Scaling a Graphics Object

An application can scale the flag by 0.5, by applying:

$$x' = 0.5x$$

$$y' = 0.5y$$

The original 5 points of the flag are transformed:

$$(0,0) \rightarrow (0,0)$$

$$(0,4) \rightarrow (0,2)$$

$$(0,6) \rightarrow (0,3)$$

$$(2,4) \rightarrow (1,2)$$

$$(2,6) \rightarrow (1,3)$$

Figure 17-7 shows the effect of the scaling.

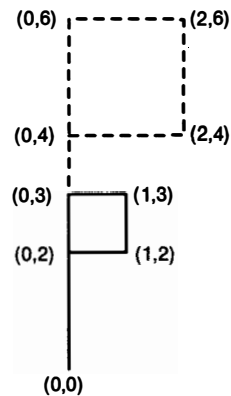


Figure 17-7. Scaling by 0.5

This scaling preserves the shape and orientation of the object, because the scaling factors in both directions are the same. However, scaling equations permit different scaling factors to be applied to x and y, which can cause distortion of the original shape of the object.

Reflecting a Graphics Object

A negative scaling factor causes a reflection of the object to be drawn. A scaling factor of -1 , for example, causes a mirror image of the object to be drawn in the appropriate direction.

Figure 17-8 shows the flag reflected by applying a negative y scaling factor.

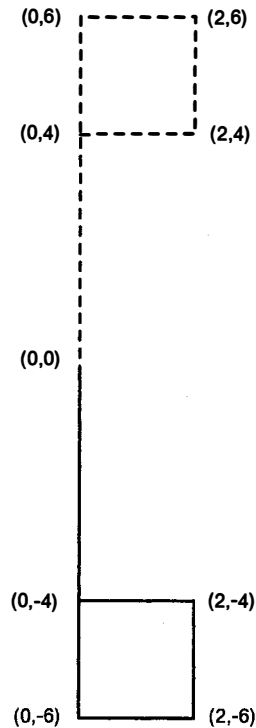


Figure 17-8. Reflection

MATRIXLF Structure for Scaling and Reflecting

When an application scales an object by using the scaling transformation, the matrix element M_{11} contains the horizontal scaling component (S_x), and the matrix element M_{22} contains the vertical scaling component (S_y).

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If the matrix element M_{11} contains a negative horizontal reflection component ($-S_x$), it causes reflection about the y-axis. If the matrix element M_{22} contains a negative vertical reflection component ($-S_y$) it causes reflection about the x-axis.

Rotation Transformations

The application can rotate an object either using GpiRotate or by modifying the MATRIXLF structure directly.

The operating system applies a transformation to all points in the source coordinate space. This means that unless an object is drawn about the origin of the source coordinate space, translation occurs when the object is rotated or scaled.

GpiRotate enables an application to specify a point, relative to the origin, that is, the center of rotation.

The equations for the rotation of an object about the origin (0,0) through an angle θ , can be written:

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

A negative θ value rotates the object clockwise. For clockwise rotation, the rotation equations are:

$$x' = x \cos \theta + y \sin \theta$$

$$y' = -x \sin \theta + y \cos \theta$$

To rotate an object about some other point (p,q), the following sequence of transformations is required:

1. Translate the object by $(-p,-q)$ to move the point of rotation to the origin.
2. Rotate the object around the origin.
3. Translate the object by (p,q) to move it back to its original position.

Rotation preserves the shape and size of the object.

Rotating a Graphics Object

An application can rotate the flag counterclockwise through 90° , by applying:

$$x' = x \cos 90 - y \sin 90$$

$$y' = x \sin 90 + y \cos 90$$

Because $\cos 90 = 0$ and $\sin 90 = 1$, these equations become:

$$x' = -y$$

$$y' = x$$

The original 5 points are transformed:

$$(0,0) \rightarrow (0,0)$$

$$(0,4) \rightarrow (-4,0)$$

$$(0,6) \rightarrow (-6,0)$$

$$(2,4) \rightarrow (-4,2)$$

$$(2,6) \rightarrow (-6,2)$$

Figure 17-9 shows the effect of rotating the flag 90°.

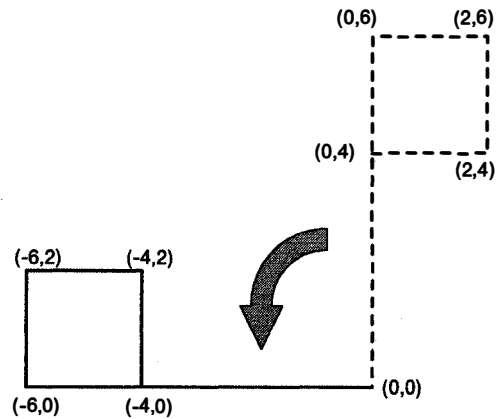


Figure 17-9. Rotation Counterclockwise through 90°

MATRIXLF Structure for Rotating

For counterclockwise rotation:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For clockwise rotation:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation Transformations

The application can translate an object either using GpiTranslate or by modifying the MATRIXLF structure directly.

To move a graphics object by an absolute number of coordinate units, a translation equation is applied. The equations for translation by T_x and T_y are obtained from the general equations (with $E = T_x$, and $F = T_y$) and can be written:

$$x' = x + T_x$$

$$y' = y + T_y$$

Translation preserves the shape, size, and orientation of the object. A negative T_x value causes movement to the left. A negative T_y value causes movement downward.

Translating a Graphics Object

If T_x is equal to 8 and T_y is equal to 5, then:

$$x' = x + 8$$

$$y' = y + 5$$

The original 5 points are transformed:

$$(0,0) \rightarrow (8,5)$$

$$(0,4) \rightarrow (8,9)$$

$$(0,6) \rightarrow (8,11)$$

$$(2,4) \rightarrow (10,9)$$

$$(2,6) \rightarrow (10,11)$$

Figure 17-10 shows the effect of translating the flag by (8,5).

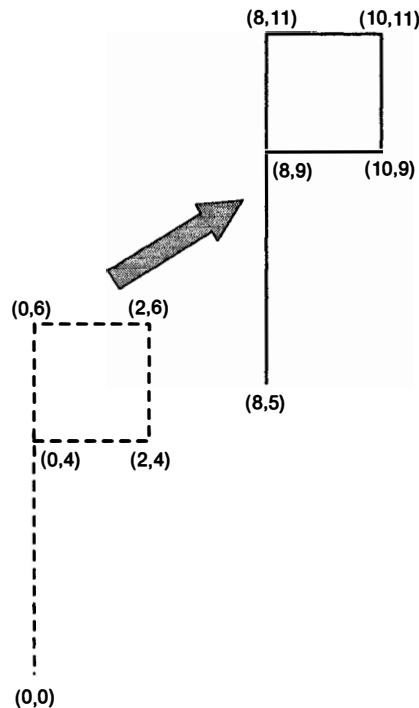


Figure 17-10. Translation by (8,5)

MATRIXLF Structure for Translating

When an application translates an object by using the translation transformation, the matrix element M_{31} contains the horizontal translation component, and the matrix element M_{32} contains the vertical translation component, as follows:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

Shearing Transformations

There are 2 shear transformations: vertical and horizontal. The vertical shear transformation affects only the y-component of the coordinates of points in an object; the horizontal shear transformation affects only the x-component.

If an application shears an object that contains 2 orthogonal vectors (2 perpendicular lines), the vectors are no longer orthogonal.

A shearing transformation alters the shape of an object by translating its x-coordinates relative to its y-coordinates, or its y-coordinates relative to its x-coordinates. The amount by which the coordinates are translated is determined by the angle of the shear.

The equation for shearing an object to the left along the x axis by angle θ is:

$$x' = x - y \tan \theta$$

$$y' = y$$

To shear an object along the y-axis, the tangent of the angle of the shear is represented by constant B in the general equation.

Shearing a Graphics Object

Figure 17-11 shows the flag sheared to the left along the x-axis.

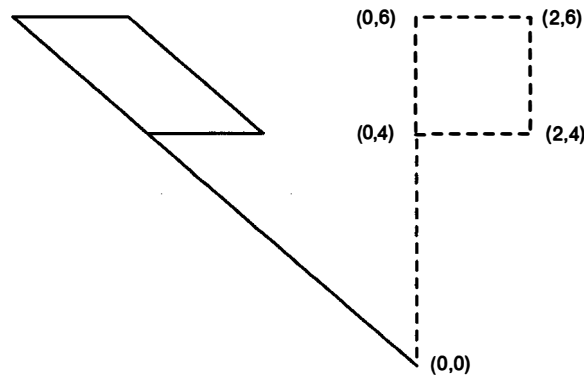


Figure 17-11. Shearing Along the X-Axis. In this example, the y-coordinates are unaltered.

MATRIXLF Structure for Shearing

For vertical shear transformation, the matrix element M_{11} contains the horizontal shear component, and the element M_{21} contains the vertical shear component, as follows:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ -\tan \theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For horizontal shear transformation, the matrix element M_{21} contains the horizontal shear component, and the matrix element M_{22} contains the vertical shear component, as follows:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} + \tan \theta & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

About Transformation Functions

Transformation functions manipulate objects between coordinate spaces by applying transformations. Transformation functions require 2 coordinate spaces: a source coordinate space, and a target coordinate space. Which transformation function an application should use is determined by the 2 coordinate spaces and by the transformation effect desired.

The world-to-model space, model-to-page space, and page-to-device space transformations are all actually performed as a single operation. The different coordinate spaces are conceptual in nature, rather than explicitly defined entities. Describing them separately is meant to help explain the levels of activity during transformations.

All transformation functions share certain parameters, although the values and defaults of those parameters might not be identical.

Current Transformation

Every graphics object, whether in world, model, page, or device space, has a *current transformation* value, even if that value is simply the *identity transformation*.

The default current model transformation is the concatenation of any instance, segment, and model transformations from the root segment downward. The default current viewing transformation is the 1 most recently specified. The device transformation, which is set by the page viewport and the presentation page, should not be changed while drawing is in progress.

Accumulating Transformations

Each time 1 of the transformation functions is called by an application, the application can set the function's option parameter to control how the function combines the transformation with existing transformations and in what order the transformations are applied.

If the application uses this flag...	Then the operating system...
TRANSFORM_REPLACE	Replaces any existing transformations with the new transformation. The existing value of the matrix is discarded and replaced by straight substitution.
TRANSFORM_PREEMPT	Applies the new transformation before applying the existing transformation. The transformation matrix of the new transformation is pre-multiplied with the transformation matrix of the current transformation.
TRANSFORM_ADD	Applies the new transformation after applying the existing transformation. The transformation matrix of the new transformation is post-multiplied with the transformation matrix of the current transformation.

The order in which transformations are applied affects the appearance of the picture. For example, suppose that a box primitive has been defined, with its lower-left corner at (4,2) and its upper-right corner at (8,8), and that you want both to scale the box by 0.5 and to translate it by $(-10,-10)$.

If the box is translated before scaling it, the transformed box is as shown in Figure 17-12.

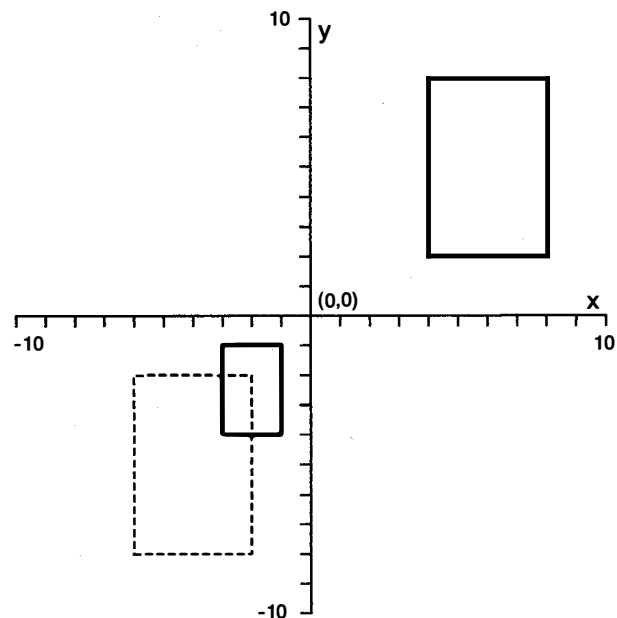


Figure 17-12. Translating before Scaling. The translated box has its lower-left corner at $(-6,-8)$, and its upper-right corner at $(-2,-2)$. Each of its coordinates is then scaled by 0.5, and the transformed box has its corners at $(-3,-1)$, $(-1,-1)$, $(-3,-4)$, and $(-1,-4)$.

If the box is scaled before translating it, the transformed box is as shown in Figure 17-13.

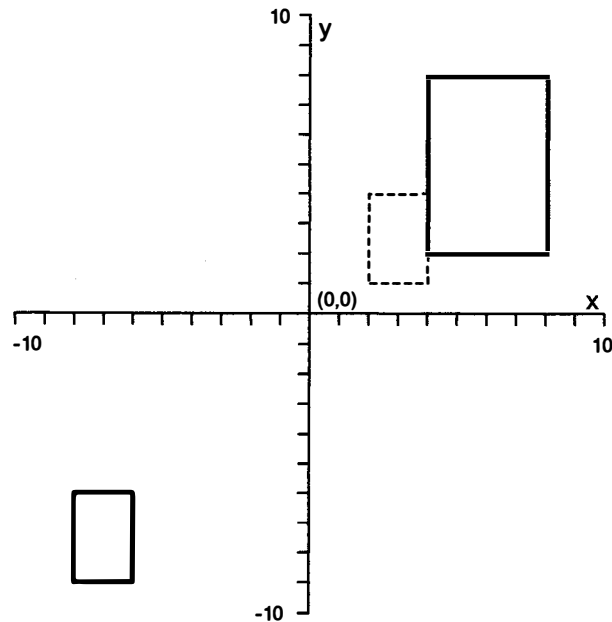


Figure 17-13. Scaling before Translating. The scaled box has its lower-left corner at (2,1), and its upper-right corner at (4,4). The box is then translated by (-10,-10), and the transformed box has its corners at (-8,-6), (-6,-6), (-6,-9), and (-8,-9).

When an application is drawing a picture in which there are called segments, and in which transformations are applied to the root segments, the root-segment transformations should usually be applied to any segments they call. For example, if a segment that is translated to the left of the picture (by changing its segment transformation) calls a second segment, that leftward transformation should also be applied to the called segment. In this instance, the application would specify `TRANSFORM_ADD` in the call to `GpiCallSegmentMatrix` to add the instance transformation to the calling segments' segment transformation. Instance transformations are automatically reset on return to the calling segment.

Concatenating Transformations

When an application applies more than 1 transformation, it can concatenate the individual transformations to produce a final result. To concatenate transformations, multiply the individual transformation matrixes. The product of this multiplication is the *concatenated transformation*.

There are four ways the final matrix can be concatenated:

1. Hard code the matrix values into the application, then call the transformation functions.
2. Multiply the individual matrixes, then call the transformation functions.
3. Use the helper functions with the `TRANSFORM_ADD` option, then call the transformation functions.
4. Alternately apply transformation operations directly to the transformation matrix, then apply a transformation function.

Concatenating before calling provides better performance.

Hard Coding Values for a Concatenated Matrix

Pre-calculating the concatenated matrix values, then hard coding the values into the application, gives the fastest performance. However, this is rarely practical as the actual transformations to be performed are often variable.

If an application is, for example, performing interactive graphics, and 1 of the options offered to the user is a menu choice of rotate by 90°, and enlarge by 4, then the rotational and scaling factors would not change, and the matrix values could be hard coded into the application.

Multiplying Matrix Values

Multiplying transformation matrix values directly offers the second fastest performance, yet can still respond to a variety of desired transformations. As many transformation matrixes as needed can be multiplied together. If the application is concatenating the matrixes itself, it is responsible for preventing the accumulated transformation side effects.

For example, to rotate an object counterclockwise about the point (p,q) using a single transformation call requires 3 transformations to be concatenated. When the application is specifying each transformation, step-by-step, the sequence of actions would be:

1. Translate the object by $(-p, -q)$ to move the point of rotation to the origin.
2. Rotate the object about the origin.
3. Translate the object by (p, q) to move it back to its original position.

The individual matrixes are:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -T_x & -T_y & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

The matrix for the concatenated transformation is produced incrementally. That is, 2 adjacent matrixes are multiplied to produce a single matrix, which is then multiplied with the third matrix. You can begin by multiplying either the first 2 matrixes or the second 2 matrixes. If you start by multiplying matrixes 2 and 3 together, the resulting matrix is:

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

This matrix is multiplied with the first matrix to produce the matrix for rotating an object at the point (p,q):

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ a & b & 1 \end{bmatrix}$$

where:

$$a = (-T_x \cos \theta + T_y \sin \theta + T_x)$$

and

$$b = (-T_x \sin \theta - T_y \cos \theta + T_y)$$

If an application were performing the concatenation for a scaling operation, again it would have to specify the transformation step-by-step. The sequence of actions would be:

1. Translate the object's scaling point to the origin.
2. Scale the object at the origin.
3. Translate the object back to its original position.

Here are the 3 matrixes required to obtain this effect:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -T_x & -T_y & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

The matrix of the concatenated transformation is:

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ (-T_x S_x + T_x) & (-T_y S_y + T_y) & 1 \end{bmatrix}$$

Transformation Helper Functions

Three helper functions, are provided to perform the matrix math required to concatenate transformations:

- GpiTranslate
- GpiRotate
- GpiScale.

Any of these 3 functions can be used with the TRANSFORM_ADD option to concatenate the new matrix with an existing matrix. This method builds up the matrix in application storage in a sequence of steps before issuing a single transform function.

While this is slower than hard coding the matrix, it is faster than alternating between applying transformation operations directly to the matrix then applying a transformation function.

The helper functions merely calculate the appropriate matrix. The transformation is not applied until the array containing the matrix values is passed to the appropriate transformation function.

Applications use GpiTranslate to change the position of an object. The application specifies the coordinates of the point to which to move the object and the name of the transform matrix to use as input to GpiTranslate. The transformation matrix must be in the form of a one-dimensional array. The application also can specify whether this transformation is to replace the value for a previous transformation, or whether it is to be added to it.

Applications use GpiRotate to rotate an object. The application specifies the angle of rotation, the coordinates of the point around which the object is to rotate, and the transformation matrix. The transformation matrix must be in the form of a 1 dimensional array. The application also can specify whether this transformation is to replace the value for a previous transformation, or whether it is to be added to it.

To scale an object at a point without also moving the object, applications use `GpiScale`. When using `GpiScale`, the application specifies the scaling factor, the coordinates of the center point, and the transformation matrix. The transformation matrix must be in the form of a one-dimensional array. The application also can specify whether this transformation is to replace the value for a previous transformation, or whether it is to be added to it.

Applying Transformation Operations Directly to the Transformation Matrix and Using Transformation Functions

An application could alternate between applying transformation operations directly to the transformation matrix, then applying a transformation function with the `TRANSFORM_ADD` option set. This would build up the matrix in the presentation space. This method has the slowest performance of the four concatenation methods.

Round-Off Error

Whenever an application uses transformations, it should handle any round-off error that occurs after multiple scaling, rotation, shear, or reflection transformations. The rounding error increases because a transformation is incrementally updated by the amount of the error with, for example, the `TRANSFORM_ADD` option. For the rotation to remain accurate, the application should recalculate the transformation, rather than accumulate many small changes.

For example, if an application uses a rotation transformation to rotate the hands of a clock, the accuracy of the clock diminishes due to rounding off after the transformation. The rounding error should be periodically removed by using the `TRANSFORM_REPLACE` option at known points, for example, every 90° or every complete revolution of the clock.

World Space-to-Model Space Transformations

The model transformation drawing attribute operates between world and model space. This attribute can be updated by 1 of the 3 following transformation functions:

- `GpiSetModelTransformMatrix`
- `GpiSetSegmentTransformationsMatrix`
- `GpiCallSegmentMatrix`.

If the model transformation drawing attribute has never been updated, the attribute defaults to the identity transformation. If the drawing attribute has been updated, the *existing transformation* is the concatenation of any instance, segment, or model transformations from the root segment downwards. Each time a new segment is opened (using `GpiOpenSegment`), the model transformation drawing attribute is reset to its default value.

The 3 transformations that operate between world space and model space are:

- Model transformations
- Segment transformations
- Instance transformations.

The transformations that occur between world-coordinate and model spaces depend on the drawing mode and the possible segment type of the drawing primitive.

The drawing mode can be either nonretain or retain. Nonretain mode is also called draw mode, as the graphics are immediately displayed. In retain mode, the

graphics orders are stored in chained and unchained segments. A series of segments are shown in Figure 17-14.

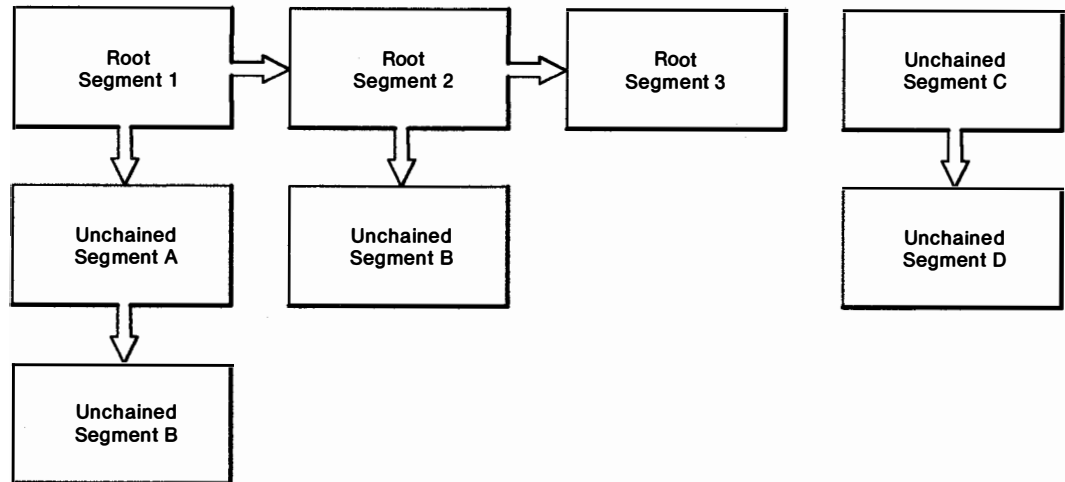


Figure 17-14. Segments. A model transformation effects objects in any of these segments. A segment transformation effects the 6 chained segments, on the left and must be issued before Root Segment 1 is accessed. An instance transformation can be applied only to Segment B, or Segment D; and must be issued from Segment A or Segment 2, or Segment C. The instance transformation is reset on return to the calling segment.

Model Transformations

Model transformations also can be applied to objects both inside and outside of segments. Model transformations also can be applied to objects created under retained or nonretained drawing mode. This means model transformations affect the output from:

- GpiDrawChain
- GpiDrawFrom
- GpiDrawSegment
- Any GPI functions that occur outside of a retained-drawing segment.

Applications can change the model transformation any number of times in a single segment. Changing the model transformation changes the transformation applied to any drawing primitives issued subsequently.

An application can determine the values for the current model transformation by calling GpiQueryModelTransformMatrix, which returns the model transformation's scaling, rotation, and translation values in a one-dimensional array representing elements in the MATRIXLF structure.

GpiSetModelTransformMatrix specifies the model transformation's scaling, rotation, and translation values applied to subsequent primitives in the segment. As this function does not require the name a segment, it also can be used to apply transformations to primitives outside of segments.

For example, in the building example earlier in the chapter, if the window is stored in a retained segment, an application could draw several identical windows on the house by setting a new translation component in the model transformation before calling GpiBox.

Segment Transformations

The segment transformation provides a way of defining the model transformation drawing attribute at the start of a retained segment. The attribute can subsequently be updated by the model transform orders within the segment. A segment transformation can be applied only to a retained segment.

Segment transformations alter retained-drawing output. Unlike a model transformation, which can be set and reset within a segment bracket, a segment transformation must be set outside of a segment bracket.

An application can determine the values for the current segment transformation by calling `GpiQuerySegmentTransformMatrix`, which returns the model transformation's scaling, rotation, and translation values in a one-dimensional array representing elements in the `MATRIXLF` structure.

To apply a segment transformation, applications use `GpiSetSegmentTransformMatrix`. `GpiSetSegmentTransformMatrix` can be used to apply any sort of transformation to a segment. For example, it can be used to translate a dynamic segment from 1 screen position to another, when movement is requested by the user. The application could perform the following steps, for example, on receipt of a `WM_MOUSEMOVE` message:

1. Use the new mouse position to calculate the required displacement from the current position.
2. Call `GpiRemoveDynamics` to remove the dynamic segment from its current position.
3. Call `GpiSetSegmentTransformMatrix` to translate the segment coordinates by the required amount.
4. Call `GpiDrawDynamics` to redraw the segment in its new position.

Instance Transformations

Instance transformations provide a way of defining the model transformation drawing attribute for the duration of a called segment.

Instance transformations alter the retained-drawing output from special segments referred to as *called* segments. A called segment usually contains a subpicture duplicated several times in other subpictures. The instance transformation positions, sizes, and rotates the subpicture each time the segment is duplicated, because the transformation is set each time the segment is called. An instance transformation applies only to the called segment, and is reset on return to the calling segment. There is no query function associated with the transformation as it must be explicitly set for each called segment.

To apply an instance transformation, applications call `GpiCallSegmentMatrix` from the calling segment. `GpiCallSegmentMatrix` calls the segment and also applies the model transformation's scaling, rotation, and translation values in a one-dimensional array representing elements in the `MATRIXLF` structure and passes a segment identifier.

World Space-to-Model Space Transformation Summary

Table 17-6 summarizes the choices available during world-coordinate space to model space transformations.

Table 17-6. World to Model Space Transformation Summary		
Transformation Type	Transformation Function	Applies to...
Model transformations	GpiSetModelTransformMatrix	Primitives both inside and outside a segment.
Segment transformations	GpiSetSegmentTransformationsMatrix	A retained segment, issued at the first element of the segment.
Instance transformations	GpiCallSegmentMatrix	Only the called segment.
Note: The effect of GpiSetModelTransformMatrix and GpiCallSegmentMatrix on the model transformation drawing attribute can be duplicated by an equivalent drawing order in a drawn retained segment.		

Model Space-to-Page Space Transformations

There are 2 transformations that operate between the model space and the page space:

- Viewing transformations
- Default viewing transformations

All 3 model transformation types and the viewing transformations can be considered a part of the picture. The default viewing transformation is part of the environment, and must not be used for picture construction.

Viewing transformations only apply in retained or nonretained segments. The viewing transformation attribute is set to the presentation space viewing transformation matrix value at the start of each drawn root segment and remains constant for that segment. If GpiSetViewingTransformMatrix is called, the new value is not used until the next segment is opened. The matrix drawing attribute is reset to identity at the end of the segment. Each change in a viewing transformation is equivalent to defining a new model space.

The default viewing transformation is a presentation space attribute, and should not normally be modified in the middle of a picture. This attribute can be updated by the transformation function GpiSetDefaultViewMatrix.

A picture is normally constructed in the presentation page with an identity default viewing transformation. The default viewing transformation can then be used to scale and scroll the entire picture in the presentation page.

Viewing Transformations

More than 1 copy of an entire model space can be drawn in page coordinate space, and each copy can be transformed as required. Parts of the model space also can be transformed to the presentation page. For example, an enlarged view of the tail of an aircraft can be shown with a reduced view of the complete aircraft in 1 corner. This is shown in Figure 17-15 on page 17-31. In this example, the picture assembled in the presentation page is derived from a single model space.

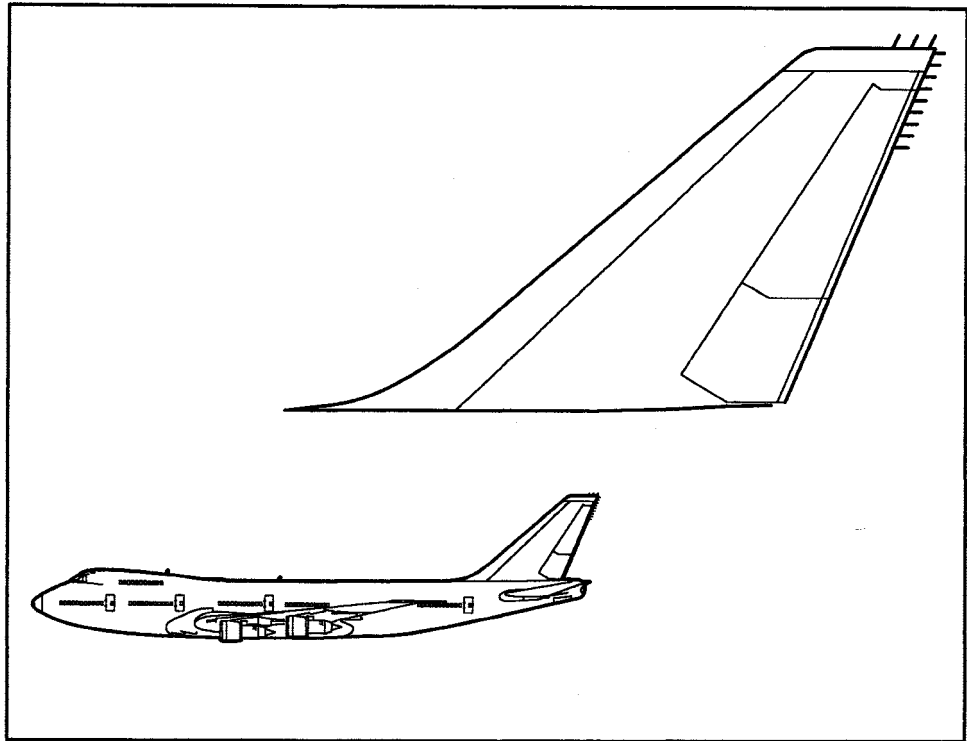


Figure 17-15. Presentation-Page Space. The entire model space (the aircraft) and a part of the model space (the tail of the aircraft) are drawn to a single page coordinate space. In each instance, scaling and translation transformations have been applied.

Alternatively, the displayed picture can be made up of several subpictures with no common graphical elements. For example, the aircraft can be drawn in 1 part of the display, and a map of an airport in another part of the display. In this instance, the final picture would be derived from different model spaces.

Whether multiple views are derived from a single model space or from different model spaces, there are 2 items to address for each instance of a model space incorporated into the presentation page:

1. The part of the model space to be displayed must be identified by defining a *viewing window* for the model space.
2. To position and size the contents of the model space in page coordinate space, a viewing transformation must be specified.

To get views of 1 or more model spaces on the screen simultaneously, each model space is drawn the required number of times. Before each drawing request, the viewing window is defined and a viewing transformation is specified.

Defining the Viewing Window

The viewing window is a conceptual boundary around a part of the model space. To produce the picture in Figure 17-15, the aircraft is drawn twice. The first time, the viewing window is on only the tail of the aircraft, and the second time, the viewing window is on the entire aircraft. Only those parts of the model space within the viewing window are visible in the picture assembled in presentation-page space. `GpiSetViewingLimits` is used to define the viewing window.

Graphics Field

Applications also can specify a type of viewing window for the presentation page smaller than the page. This window is known as the *graphics field*. To define a graphics field use `GpiSetGraphicsField`. By default, no graphics field is specified. If a graphics field is defined, the picture assembled within it is the picture that is visible on the output device.

An application can determine the current values for the viewing transformation by calling `GpiQueryViewingTransformMatrix`, which returns the transformation values in a one-dimensional array representing elements in the `MATRIXLF` structure. The application can set the values by calling `GpiSetViewingTransformMatrix`, and passing the transformation values in a one-dimensional array representing elements in the `MATRIXLF` structure.

Default Viewing Transformations

The default viewing window is the same size as the model space. Therefore, to display 1 or more entire model spaces, draw the picture the required number of times and let the viewing window default each time.

Default viewing transformations scroll or zoom pictures in a window on a display screen. An application can determine the current values for the default viewing transformation by calling `GpiQueryDefaultViewMatrix`, which returns the default-viewing-transformation values in a one-dimensional array representing elements in the `MATRIXLF` structure. The application can set these values by calling `GpiSetDefaultViewMatrix` and passing the transformation values in a one-dimensional array representing elements in the `MATRIXLF` structure.

A default viewing transformation is applied when the screen contents are zoomed or scrolled by user interaction. A picture is zoomed when the user wants to increase or decrease the size of an area of interest. A picture is usually scrolled when there is more in the presentation page than can be displayed in a single page of output. Anything lying off the screen, but within the range of the presentation page, can be scrolled into view. The default viewing transformation applies to the entire page coordinate space, and can be added to, or can replace, the current default viewing transformation. The PM programming interface applies it after any viewing transformations.

When a presentation page is created, the default viewing transformation is set to identity. For example, if the presentation-page contents are scrolled:

- Erase the screen contents.
- Call `GpiSetDefaultViewMatrix` to translate the presentation-page picture by the required amount.
- Draw the picture again.

Figure 17-16 shows the airplane presentation-page contents scrolled to the left.

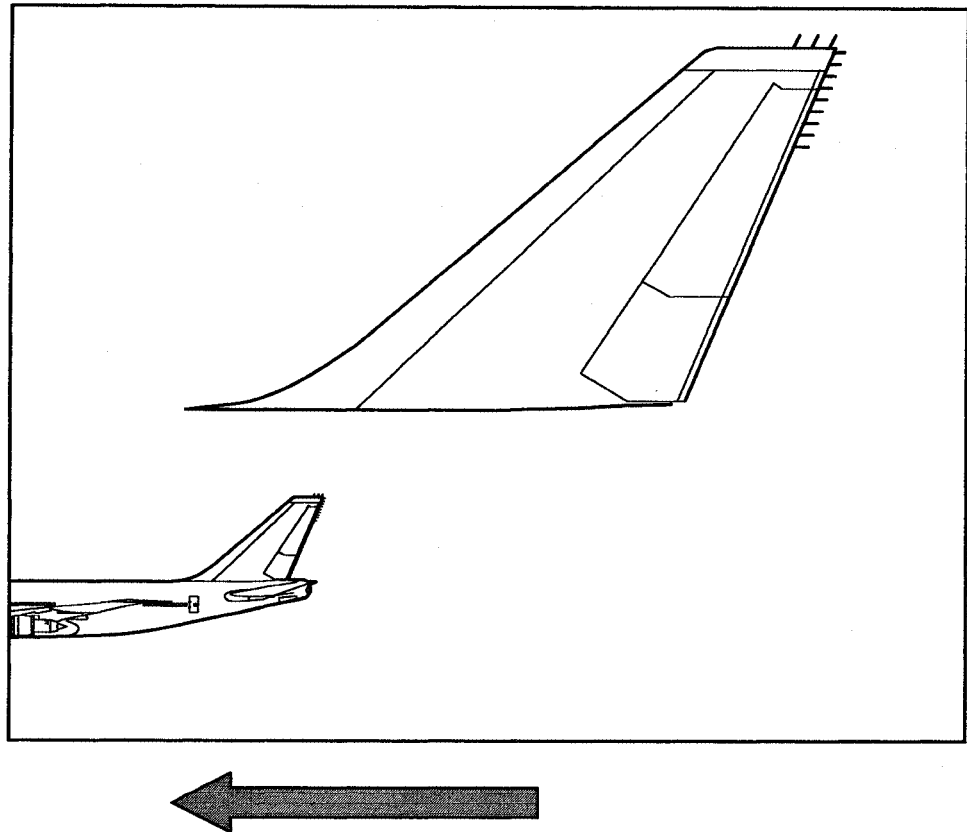


Figure 17-16. Scrolling the Presentation Page. Every presentation-page coordinate is translated to the left by the same amount.

Zooming is implemented in the same way, except that the default viewing transformation is used to scale the picture up or down as required.

If you want to display only 1 view of a single picture, and if you do not want scrolling and zooming capabilities, you can let the viewing and default viewing transformations default. When both transformations are permitted to default, page coordinate space is effectively the same as model space.

Page Space-to-Device Space Transformations

There is only 1 transformation between the page coordinate space and the device space, the *device transformation*. The device transformation enables applications to work in any presentation-page units regardless of the target device. Unlike the transformations previously described, the device transformation:

- Only scales and translates objects
- Is defined by 2 rectangles.

The first rectangle is called the *presentation page*. Its location is the bottom left origin and its dimensions are fixed. The second is called the *page viewport*. Its location is the bottom left origin and its dimensions can be varied.

The device transformation, which maps the picture in presentation-page space to device space, happens automatically. The device transformation is established when the presentation space is created, and ensures that graphics are displayed in the correct size and, where possible, that their aspect ratio is preserved. The rules

by which the device transformation is implemented are described in “Mapping the Presentation Page to the Device” on page 17-34.

To modify the device transformation, applications use `GpiSetPageViewport`. Input for this function is the device coordinates of the lower-left and upper-right corners of the page viewport. Applications should modify the default device transformation only when it is necessary to use nonstandard page units.

Presentation Pages

A presentation page is a rectangle in a page space. Its lower-left corner is always positioned at the origin of the page space.

An application can determine the dimensions of the presentation page by calling `GpiQueryPS`. It returns a pointer to a `SIZEL` structure that contains the page dimensions. If an application specifies arbitrary page units when creating a presentation space with `GpiCreatePS`, the page viewport is constructed such that the origin of the page rectangle maps to the origin of the default device rectangle and either the right or top edge maps to the corresponding edge. Thus, the aspect ratio of the graphic is preserved.

If either the height or width of the presentation page is set to 0 (using `GpiCreatePS`), the application must set `GPIA_ASSOC` to set the default presentation page size to the default device rectangle size.

Page Viewports

A page viewport is a rectangle in device space, whose origin and size can be varied. The operating system uses the presentation-page rectangle and the page-viewport rectangle to define the device transformation.

An application can determine the current dimensions of the page viewport by calling `GpiQueryPageViewport`, which returns a pointer to a `RECTL` structure that contains the coordinates of the viewport. The application can set the location and dimensions of the page viewport by calling `GpiSetPageViewport` and passing it a pointer to a `RECTL` structure that contains the new values.

The ratio of the page width to the page-viewport width defines a horizontal scaling factor. The ratio of the page height to the viewport height defines a vertical scaling factor. Applications can use `DevQueryCaps` to obtain the horizontal and vertical resolution of a device in pels per meter. The dimensions of a pel can vary from 1 device to another, but they usually fall in the range of 0.25 to 0.50 mm.

The page viewport can be shifted in the device space by a translation transformation.

Mapping the Presentation Page to the Device

When an application associates a presentation space with a device context, a default device transformation is set. A page viewport is defined according to the rules in the following table:

Presentation-page specification	Page viewport size	Usage.
Pels	The same size as the presentation page.	The lower-left corner of the presentation page maps to the lower-left corner of the device space. For example, if an application defines a presentation page of 300 coordinates (x-axis)-by-200 coordinates (y-axis), then the picture is transformed to a screen area of the same size.
Metric units	The coordinates that produce the correct matrix for the physical spacing of the pels.	The lower-left corner of the presentation page maps to the lower-left corner of the device space.
Arbitrary units	The default size for the device is used. For a plotter or printer, this is the maximum accessible area of the paper, and for a screen, it is the maximized window size.	The page viewport is constructed such that the presentation-page coordinates give equal x- and y-spacing. The lower-left corner of the presentation page maps to the lower-left corner of the device space, and either the right or the top edges map, such that the picture is contained within the device rectangle and its aspect ratio is preserved.

Figure 17-17 shows mapping from the presentation page to the device.

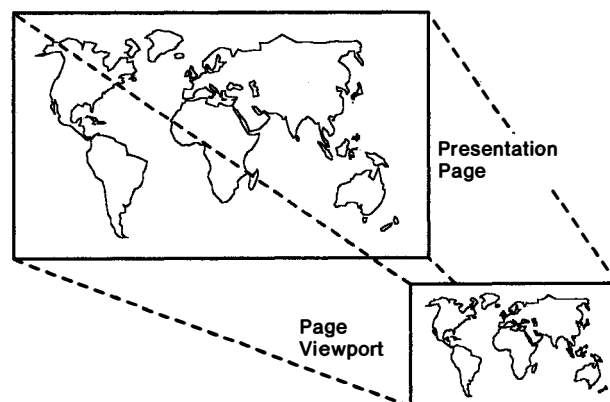


Figure 17-17. Mapping a Picture from the Presentation Page to the Device. In this example, a map of the world has been drawn in a presentation page, defined in arbitrary units, that is much larger than the device space. The device transformation scales the picture to fit the maximized window size and preserves its aspect ratio.

The device transformation can be explicitly specified using `GpiSetPageViewport`.

Coding the Device Transformation

The PM programming interface automatically transforms the presentation-page contents to the area of the device space within the page viewport. The drawing is not clipped to the page viewport because this is a scaling transformation only. The entire picture is displayed, regardless of the size of the page viewport specified. Figure 17-18 shows the airplane presentation-page contents scaled to fit the page viewport.

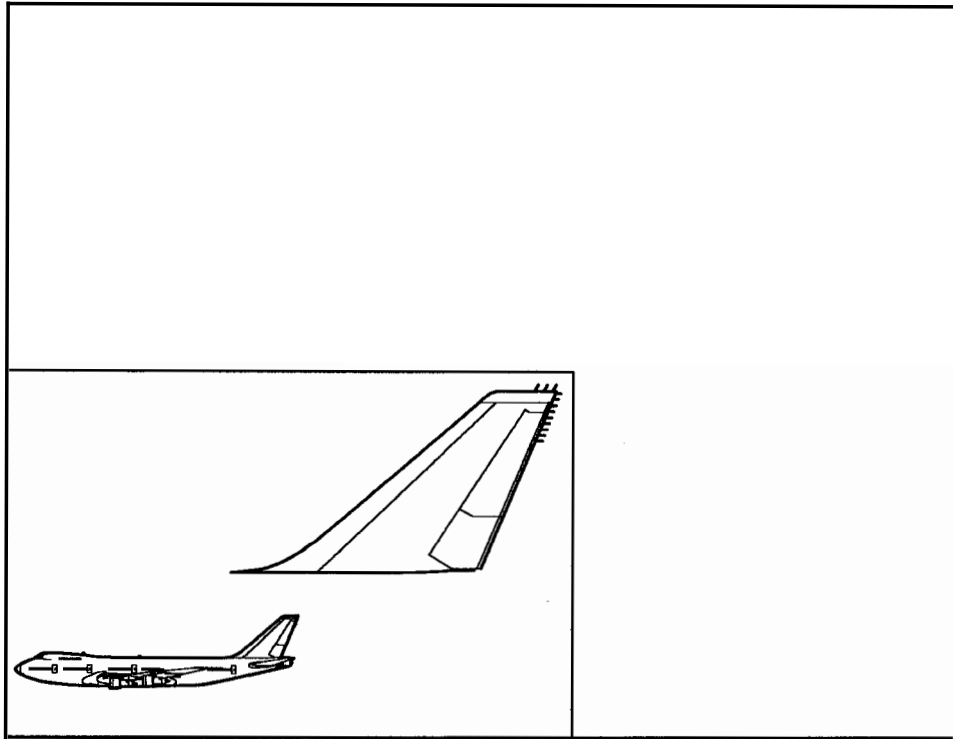


Figure 17-18. Device Space. A page viewport smaller than the presentation page has been defined. The picture assembled in the presentation page is therefore scaled to fit the page viewport.

After transformation to device space, graphics coordinates must be in the range -32768 through $+32767$, even if the presentation page is defined in GPIF_LONG format. An attempt to address a coordinate outside this range results in a coordinate-overflow error. To determine if a graphics object will give an error, applications can do the following:

- If the application is not rotating or shearing a graphics object, it calls GpiConvert to convert the device-space limits to world-coordinate-space limits, then uses these limits when creating the graphics object.
- If the application is rotating or shearing a graphics object, it uses GpiConvert to convert the device-space limits back to model space, and ensures that the picture boundary is inside these limits. Note that this method only applies if all rotational and shearing is performed using 1 of the model transformation types.

Remember that world-coordinate space has its own limits:

- -32768 through $+32767$ for a GPIF_SHORT-format presentation page
- -134217728 through $+134217727$ for a GPIF_LONG-format presentation page.

Although applications can specify a page viewport of any size, the presentation page can be mapped only to an area equal to, or less than, the available device

space. If an application specifies a viewport larger than the available device space, part of the presentation page contents are displayed outside the visible device output area. To find out the dimensions of the page viewport for the currently associated device, use `GpiQueryPageViewport`. Applications can store the dimensions of the current page viewport before changing them, and restore them later.

Device-Transformation Matrix

To directly manipulate the device-transformation matrix it is necessary for applications to know the following values:

- X_1 the x-coordinate of the lower-left corner of the page viewport
- Y_1 the y-coordinate of the lower-left corner of the page viewport
- X_2 the x-coordinate of the upper-right corner of the page viewport
- Y_2 the y-coordinate of the upper-right corner of the page viewport
- X_{PS} the presentation-space width -1
- Y_{PS} the presentation-space height -1 .

The device-transformation matrix is defined as:

The device-transformation matrix: $\begin{bmatrix} M_{11} & 0 & 0 \\ 0 & M_{22} & 0 \\ M_{31} & M_{32} & 1 \end{bmatrix}$	where: $a = (X_2 - X_1) / (X_{PS} + 1)$ $b = (Y_2 - Y_1) / (Y_{PS} + 1)$ $c = X_1 + (a - 1) / 2$ $d = Y_1 + (b - 1) / 2$
--	---

Windowing-System Transformation

There is 1 transformation, the *windowing-system transformation*, performed automatically by the PM programming interface. The windowing-system transformation, which is a translation transformation only, maps the coordinates of the picture in device space to the coordinates of the screen window or printer page. This happens when a picture is first drawn, and whenever the display window in which the picture has been drawn is moved.

Transforming Bit-Map Data

In general, graphics defined in device coordinates (bit maps and image primitives) cannot be transformed. For example, the size of an image primitive is specified in device coordinates, and cannot be altered. The size, therefore, remains unaltered down the viewing pipeline. The position of an image primitive, however, is specified in world coordinates. The image is therefore subject to translation transformations. Note, however, that `GpiWCBitBlt` permits the target rectangle to be specified in world coordinates, which are transformed.

Because the position of the image primitive is specified in world coordinates and its width is specified in device coordinates, positioning 2 images together on the screen causes special difficulties. The second image cannot be positioned without knowing the width, in world coordinates, of the first image. To get the width of the first image:

1. Identify 2 coordinate positions, 1 on the image's left edge, and 1 on its right edge. For example, the 2 positions could be (10,80) and (150,80). These positions are in device coordinates.
2. Convert these device coordinates to world coordinates using GpiConvert. GpiConvert converts an array of (x,y) coordinates that apply in 1 coordinate space to their corresponding values in another coordinate space.
3. Subtract the lower x-coordinate from the higher x-coordinate. In the above example, the width of the image is the difference between the world-coordinate equivalents of 150 and 10.

When you have the width of the first image in world coordinates, you can calculate the start position of the second image.

Paths, although defined in world coordinates, are device-dependent and are bound in device coordinates when they are defined. Subsequent transformations (other than the windowing-system transformation) have no effect on paths. However, if a path is used to create a wide line, the width of the line is scaled as required.

Using Coordinate Spaces and Transformations

This section explains how to:

- Set an application's drawing units to convenient units
- Translate, rotate, and scale a picture
- Shear a picture.

Setting Drawing Units

Applications can use `GpiCreatePS` to set the device transformation so that it uses page units that might be more convenient than pels; for example, centimeters. If the output device is a screen, the application first opens a device context by calling `WinOpenWindowDC`. If the output device is a printer or plotter, the application opens a printer or plotter device context by calling `DevOpenDC`. The application then creates a presentation space by calling `GpiCreatePS`, specifying low-metric page units and associating the device context with the presentation space. Figure 17-19 is an example of how to set drawing units.

```
HWND hwnd;           /* Client-window handle */
HAB hab;             /* Anchor-block handle */
HPS hps;             /* Presentation-space handle */
HDC hdc;             /* Device-context handle */
SIZEL sizlPage;      /* Presentation-page rectangle */

hdc = WinOpenWindowDC(hwnd);
sizlPage.cx = 0;
sizlPage.cy = 0;
hps = GpiCreatePS(hab, /* Anchor-block handle */
                 hdc,  /* Device-context handle */
                 &sizlPage, /* Address of SIZEL structure */
                 PU_LOMETRIC /* Centimeters as page units */
                 | GPIA_ASSOC); /* Associates window DC with PS */
```

Figure 17-19. Setting Drawing Units

Translating, Rotating, and Scaling a Picture

`GpiTranslate`, `GpiRotate`, and `GpiScale` provide a convenient method for transforming objects in a picture. Figure 17-20 on page 17-40 shows how to use these functions to translate, rotate, and scale a triangle.

```

MATRIXLF matlTransform;
POINTL ptlStart, ptlTrans, ptlRotate, ptlScale;
FIXED fxAngle, afxScale[2];
POINTL aptlTriangle[] = { 575, 300, 575, 500, 500, 300 };

ptlStart.x = 500; /* Starting point x direction */
ptlStart.y = 300; /* Starting point y direction */
GpiMove(hps, &ptlStart);
GpiPolyLine(hps, sizeof(aptlTriangle) / sizeof(POINTL), aptlTriangle);

ptlTrans.x = 75; /* x coordinate for translation */
ptlTrans.y = 75; /* y coordinate for translation */

GpiTranslate(hps, /* Presentation-space handle */
             &matlTransform, /* Address of matrix */
             TRANSFORM_REPLACE, /* Replace old matrix with new */
             &ptlTrans); /* Coordinates for translation */

GpiSetModelTransformMatrix(hps, /* Presentation-space handle */
                           9, /* Number of points in matrix */
                           &matlTransform, /* Address of matrix */
                           TRANSFORM_REPLACE); /* Replace old matrix with new */

GpiMove(hps, &ptlStart); /* Move to starting point */
GpiPolyLine(hps, sizeof(aptlTriangle) / sizeof(POINTL), aptlTriangle);

fxAngle = MAKEFIXED(-45, 0); /* Rotate 45 degrees clockwise */
ptlRotate.x = 550; /* x coordinate rotation origin */
ptlRotate.y = 350; /* y coordinate rotation origin */

GpiRotate(hps, /* Presentation-space handle */
          &matlTransform, /* Address of matrix */
          TRANSFORM_REPLACE, /* Replace old matrix with new */
          fxAngle, /* Rotation angle */
          &ptlRotate); /* Origin of rotation */

GpiSetModelTransformMatrix(hps, 9, &matlTransform, TRANSFORM_REPLACE);

GpiMove(hps, &ptlStart); /* Move to starting point */
GpiPolyLine(hps, sizeof(aptlTriangle) / sizeof(POINTL), aptlTriangle);

ptlScale.x = 550; /* x coordinate scale origin */
ptlScale.y = 350; /* y coordinate scale origin */
afxScale[0] = MAKEFIXED(2, 0); /* Scaling factor on x axis */
afxScale[1] = MAKEFIXED(2, 0); /* Scaling factor on y axis */

GpiScale(hps, /* Presentation-space handle */
         &matlTransform, /* Address of matrix */
         TRANSFORM_REPLACE, /* Replace old matrix with new */
         &afxScale[0], /* Scaling factor */
         &ptlScale); /* Origin of scaling operation */

GpiSetModelTransformMatrix(hps, 9, &matlTransform, TRANSFORM_REPLACE);

GpiMove(hps, &ptlStart); /* Move to starting point */
GpiPolyLine(hps, sizeof(aptlTriangle) / sizeof(POINTL), aptlTriangle);

```

Figure 17-20. Translating, Rotating, and Scaling a Triangle

Shearing a Picture

Figure 17-21 is an example of shearing a picture by modifying the transformation matrix directly.

```
MATRIXLF matlfTransform;
POINTL ptlStart, ptlEnd;

ptlStart.x = 500;          /* x coordinate, lower-left corner of box */
ptlStart.y = 300;          /* y coordinate, lower-left corner of box */
GpiMove(hps, &ptlStart);
ptlEnd.x = 700;            /* x coordinate, upper-right corner of box */
ptlEnd.y = 500;            /* y coordinate, upper-right corner of box */
GpiBox(hps, DRO_OUTLINE, &ptlEnd, 0, 0); /* Draw first box */

matlfTransform.fxM11 = MAKEFIXED(1, 0);
matlfTransform.fxM12 = MAKEFIXED(0, 0);
matlfTransform.lM13 = 0;
matlfTransform.fxM21 = MAKEFIXED(0, 65536 / 2); /* Shear factor .5 */
matlfTransform.fxM22 = MAKEFIXED(1, 0);
matlfTransform.lM23 = 0;
matlfTransform.lM31 = 200; /* Translate 200 units right */
matlfTransform.lM32 = 0;
matlfTransform.lM33 = 1;
GpiSetDefaultViewMatrix(hps, 9, &matlfTransform, TRANSFORM_REPLACE);

GpiMove(hps, &ptlStart);
GpiBox(hps, DRO_OUTLINE, &ptlEnd, 0, 0); /* Draw sheared box */
```

Figure 17-21. Shearing a Picture

Using World to Model Space Transformations

Figure 17-22 is an example of a sequence of calls in which segment, model, and instance transformations are applied to graphics objects.

```
GpiSetDrawingMode (DM_RETAIN) /* Sets the current drawing mode */
/* to DM_RETAIN */

GpiOpenSegment (segment 1) /* Creates a chained segment */

GpiCloseSegment
GpiSetSegmentAttrs /* Make segment 1 an unchained segment */

GpiOpenSegment (segment 2) /* Creates a retained, chained segment */
```

```

GpiSetModelTransformMatrix (TRANSFORM_ADD)
/* Specifies a transformation to */
/* apply to subsequent primitives. */
/* This is in addition to the current */
/* model transformation. */

GpiCallSegmentMatrix (1, TRANSFORM_ADD)
/* Calls segment 1 and applies a */
/* transformation to it. */
/* This transformation is in addition */
/* to the current model transformation, */
/* and applies only to the called */
/* segment. */

.
GpiSetCurrentArcParams
GpiPointArc
/* The 3-point arc is not subject */
/* to the transformation specified */
/* on the call to GpiCallSegmentMatrix. */
/* The transformation that was */
/* current before segment 1 was called */
/* is applied to the remainder of */
/* segment 2. */

GpiCloseSegment
GpiSetSegmentTransformMatrix (segment 2)
/* Specifies a segment transformation */
/* for segment 2 */

GpiDrawSegment
/* Draws segment 2 */

```

Figure 17-22 (Part 2 of 2). Using World-to-Model Transformations

Viewing Transformation

Each time an application draws the model space, it specifies a different viewing transformation to transfer a view of the picture-to-page coordinate space.

`GpiSetViewingTransformMatrix` is used to apply a viewing transformation. There is 1 viewing transformation for each part or whole model space to be incorporated in page-coordinate space.

`GpiSetViewingTransformMatrix` cannot be called while there is an open segment, and it has no effect on primitives outside segments. When it has been specified, the viewing transformation applies to all subsequently created segments until it is next changed. The viewing transformation that is current when a segment is created is a fixed part of the segment. Once specified, the viewing transformation cannot be queried, nor can it be altered, unless you re-create the segment. This is inconvenient when several versions of the same picture are being drawn. The problem can be solved by:

- Defining the picture in 1 or more unchained segments.
- Creating a segment chain and calling the unchained segments from each root segment.
- Setting the viewing transformation before each root segment.

Each time the segment chain is drawn, multiple views of the same picture are produced. This sequence is illustrated in the following example, where the segment chain comprises 3 root segments, each of which calls a single unchained segment.

GpiSetInitialSegmentAttrs	/* Switches off the chained attribute */
GpiOpenSegment	/* Creates an unchained segment */ /* containing the picture definition */
GpiCloseSegment	
GpiSetInitialSegmentAttrs	/* Switches on the chained attribute */
GpiSetViewingTransformMatrix	/* Sets the viewing transformation */ /* for segment 1 */
GpiOpenSegment (segment 1)	
GpiCallSegmentMatrix	/* Calls the unchained segment */
GpiCloseSegment	
GpiSetViewingTransformMatrix	/* Sets the viewing transformation */ /* for segment 2 */
GpiOpenSegment (segment 2)	
GpiSetViewingLimits	/* Specifies the area of interest */ /* in the model space. */
GpiCallSegmentMatrix	/* Calls the unchained segment */
GpiCloseSegment	
GpiSetViewingTransformMatrix	/* Sets the viewing transformation */ /* for segment 3 */
GpiOpenSegment (segment 3)	
GpiCallSegmentMatrix	/* Calls the unchained segment */
GpiCloseSegment	

Figure 17-23. Using Viewing Transformations

When a segment chain has been created using this method, an application cannot change the viewing transformation unless it re-creates the segment chain. The viewing transformation of a segment is permanently recorded and cannot be edited. The application would not, however, have to re-create the picture definition in the unchained segment.

If the picture definition comprises a number of unchained segments, an application must create an intermediate segment to contain the GpiCallSegmentMatrix calls for those segments. Each root segment would then call the intermediate segment.

The viewing transformation applies to the entire root segment and cannot be overridden from within the segment. It is particularly useful for positioning and scaling 1 or more segments of a subpicture within the presentation page when a segment transformation cannot be used. A segment transformation can be overridden by any model or instance transformations within the segment. A typical example of its use is when importing a subpicture using GpiPutData or GpiPlayMetaFile.

Note: The viewing transformation must be set to its default value before an application defines an unchained segment to be called from another segment.

Summary

Table 17-7 summarizes the functions used with coordinate spaces and transformations.

<i>Table 17-7. Coordinate-Space and Transformation Functions</i>	
Function Name	Description
GpiCallSegmentMatrix	Applies an instance transformation.
GpiConvert	Transforms an array of points from one coordinate space to another.
GpiConvertWithMatrix	Transforms an array of points using a specified set of transformation matrix values.
GpiCreatePS	Defines presentation page size and initial page viewport.
GpiQueryDefaultViewMatrix	Determines the current values for the default viewing transformation.
GpiQueryModelTransformMatrix	Determines the values for the current model transformation.
GpiQueryPageViewport	Determines the coordinates of the current page viewport.
GpiQuerySegmentTransformMatrix	Determines the values for the current segment transformation.
GpiQueryViewingTransformMatrix	Determines the current values for the viewing transformation.
GpiRotate	Generates a transformation matrix to perform rotation by a given number of degrees.
GpiScale	Generates a transformation matrix to scale by a given scaling factor.
GpiSetDefaultViewMatrix	Defines the default viewing transformation values.
GpiSetModelTransformMatrix	Defines the model transformation attribute values, the values to be stored in a retained segment, or both.
GpiSetPageViewport	Sets page viewport size and origin.
GpiSetSegmentTransformMatrix	Defines the segment transformation values to be stored in a specified retained segment.
GpiSetViewingLimits	Defines the viewing window.
GpiSetViewingTransformMatrix	Defines the values of the viewing transformation matrix.
GpiTranslate	Generates a transformation matrix to translate by a given horizontal value, vertical value, or both.

Table 17-8 summarizes the data structures used to manipulate coordinate spaces and transformations.

<i>Table 17-8. Coordinate-Space and Transformation Structure</i>	
Structure Name	Description
MATRIXLF	The values of the transformation matrix.
RECTL	The values of the page viewport.
SIZEL	The presentation page dimensions.

Chapter 18. Print Job Submission and Manipulation

The print subsystem of the OS/2 operating system provides a flexible, high-level interface between your application and an output device. Because most of the internal workings of the subsystem are shielded from both user and programmer, this chapter concentrates on the programming process required to produce hardcopy output. Querying of printer resources, working under a specific user's environment, designing for application-specific requirements, and manipulation of print jobs also are described.

About the Print Subsystem Components

The print subsystem comprises the following software components of the operating system:

- Spooler
- Print subsystem user interface
- Queue drivers (queue processors)
- Printer drivers
- File system
- Kernel device drivers.

Spooler

The *spooler* is the central coordinating process for the print subsystem. It gives the user flexibility in organizing and optimizing the use of the system's printers. The spooler has a global view of the system's printing resources, particularly in a server environment; and, therefore, is able to make the best use of those resources, as follows:

- Print output from two separately executing applications cannot be intermixed on the printer. As an optional optimization, the spooler can start a printing job before it is completely queued. Any successive jobs are queued normally.
- The spooler can print a job in the background while the user continues to use the application. Other single-tasking operating systems, such as DOS, require the user to wait until the print output is sent completely to the printer.
- The spooler can send jobs from PM applications, across a network, to a remote server, without the application's knowledge.

Note: The file system handles print jobs from non-PM applications.

- Queues within the spooler can be used for various purposes. For example, one queue can be used for large print jobs that are printed at times when print demand is low. Or a queue can be configured to print jobs using a special size paper; print jobs on this queue would be held until the correct paper is loaded into the printer.
- The spooler can support a number of printers simultaneously. It can be configured so that jobs on a single queue can be shared among all the printers. This load balancing, which is particularly important in server environments, can be achieved without the application's knowledge.
- Jobs can be prioritized while in the queue. For example, an urgent job can be given a higher priority than ordinary jobs.

The spooler consists of one or more print queues; one for each printer object defined by the user. Jobs are created by applications and placed in a queue, waiting to print. When the previous job is completed, the next job in the queue is sent to the printer.

Print Job Formats

The print jobs held in a spooler queue are known as *spool files*. Spool files contain the following:

- Parameters submitted with the print job
- Print job data.

Print job data is in one of two formats:

PM_Q_STD Standard output data, PM_Q_STD, is spooled as a PM metafile; that is, as a series of graphics orders stored in a packed binary format. PM_Q_STD print jobs are created through the GPI.

An advantage of the PM_Q_STD format is that the files are smaller than PM_Q_RAW format files. The smaller size saves disk space for jobs in the spooler queue and reduces network traffic when transmitting the data to a network server.

The content of PM_Q_STD jobs can be viewed using the PICVIEW application. This is achieved using the job-content menu on a print job in the printer object. The multi-page spool file can be shown in a device-independent manner so that the content of the job can be recognized easily.

After a job is spooled and ready to print, the spooler sends the PM_Q_STD job to the queue driver. The queue driver replays the metafile, through the GPI, to the appropriate printer driver. The driver, in turn, converts the data to printer-specific commands, that is, a printer-specific format.

There are some restrictions on the content of PM_Q_STD jobs that are related to the restrictions for PM metafiles. For details, refer to the *Presentation Manager Programming Reference, Volume III, Appendix G*. PM applications that cannot deal with these restrictions should enqueue print jobs using PM_Q_RAW, but there is an increase in required disk space and, possibly, network traffic.

Print jobs sent to network servers that do not support PM are converted automatically to the PM_Q_RAW format by the system. The application still can continue specifying PM_Q_STD.

Note: The effect of converting all PM_Q_STD print jobs to PM_Q_RAW can be turned on by the user's selecting **Printer-specific format** in a printer object settings page.

PM_Q_RAW Raw data, PM_Q_RAW, is the actual printer command to print the job. For example, raw data created for an HP® LaserJet® printer contains Printer Command Language (PCL) commands; and raw data created for a PostScript® printer contains PostScript commands.

The content of PM_Q_RAW jobs can be viewed with the system editor. This is achieved using the job content menu on a print job in the printer object. However, it is not always easy to recognize the content of a job. For example PostScript is very hard to understand and get a visual idea of the actual output.

Print jobs are created by the file system as a result of printing directly to the physical port using either INT 17 or INT 21 under DOS or the OS/2 DosOpen API. These print jobs always are queued using PM_Q_RAW. The actual queue chosen depends on the port used and the configuration of the print subsystem. See "Submitting a Non-Presentation Manager (Base) Print Job" on page 18-9 for more information.

PM applications also can use the PM_Q_RAW format, but the overall print-job creation process normally is slower because the printer driver has to perform more work to create the printer-specific format.

If the **Print while spooling** printer object setting is turned on, the user can perceive a faster response. In particular for a multi-page document, the first page starts printing as soon as the printer driver has completely finished converting the GPI to the printer-specific format.

Note: PM applications always should specify the PM_Q_STD format; the difference in disk space used can be from a factor of 2 to a factor of 50. The **Printer-specific format** and **Print while spooling** printer object settings can be used to configure optimal performance for the environment.

Although the base operating system supports the two print job formats described above, other formats can be supported by providing an appropriate queue driver. Refer to the *OS/2 2.0 Presentation Driver Reference*.

Disabling the Spooler

The workplace user interface to control the spooler can be found in the System Setup folder. For special circumstances or applications, the spooler object can be used to disable the spooler. Generally, this is not recommended because the output from two applications can intermix, or one application can be paused until the first application has finished sending data to the printer.

Print Subsystem User Interface

The print subsystem user interface is composed of printer objects. The spooler implements each printer object by using a queue to hold the jobs. The queue is connected to a logical device that specifies configuration data about the actual physical device, for example, the port and printer drivers. See "Print Subsystem Configuration" on page 18-4 for details.

The print subsystem user interface performs the following basic functions:

- **Print job status**

Opening a printer object folder displays an icon or detail view of the print jobs waiting in the spooler queue. Opening the settings on an individual job displays the parameters that were used when the job was queued. Some settings, such as the number of copies, can be changed while the print job is waiting in the queue.

- **Print job manipulation**

Individual print jobs can be held in or released from the queue. *Holding* a print job means that it is not printed. Individual or all the print jobs in a queue can be deleted.

- **Queue manipulation**

Queues also can be held, released, deleted, copied, or created using the printer object context menu.

- **Printer object configuration**

Opening a printer object settings notebook enables a user to browse and modify the configuration of a printer object. For example, the printer driver can be changed if the user just obtained new printer.

Queue Driver

The queue driver is also termed a *queue processor*. It is used to take print jobs from a queue and print the data using the printer driver. The print job data (either PM_Q_STD or PM_Q_RAW format) is passed through the GPI. For PM_Q_STD jobs, GpiPlayMetafile is used; and for PM_Q_RAW jobs, the DevEscape DEVESC_RAWDATA is used.

Printer Driver

Printer drivers know all details of the printer they support; therefore, printer drivers are unique for each model of printer supported by the operating system. The printer driver is responsible for:

- Displaying a dialog that enables the user to inform the system how the physical printer is configured; for example, which paper sizes are installed.
- Displaying a dialog that enables the user to configure an individual print job; for example, which orientation (portrait or landscape) to use.
- Responding to application queries for available printer capabilities such as color, resolution and forms.
- Converting the GPI commands in a print job to the printer-specific language commands that will produce the expected output. The printer-specific commands are passed to the file system.

File System

The file system is involved in both the spooling process and the printing process. When non-PM applications create print data, the file system intercepts the data and places it on a spooler queue. After a printer driver has processed a print job, the file system sends the data to the appropriate file or device using a kernel device driver.

Kernel Device Driver

The system provides device drivers for physical devices. The two most commonly used by the print subsystem are the parallel port driver and the serial port driver.

Print Subsystem Configuration

From a user's viewpoint, a printer object represents a printer. The user can specify the printer object settings for configuration; for example, which printer driver and port to use.

From a programmer's viewpoint, print configuration is more complicated. Each printer object actually consists of a queue connected to a logical device.

Figure 18-1 on page 18-5 shows some example configurations. The top-left and

top-right pictures show one printer object; the bottom-left picture shows two printer objects; and the bottom-right picture shows three printer objects.

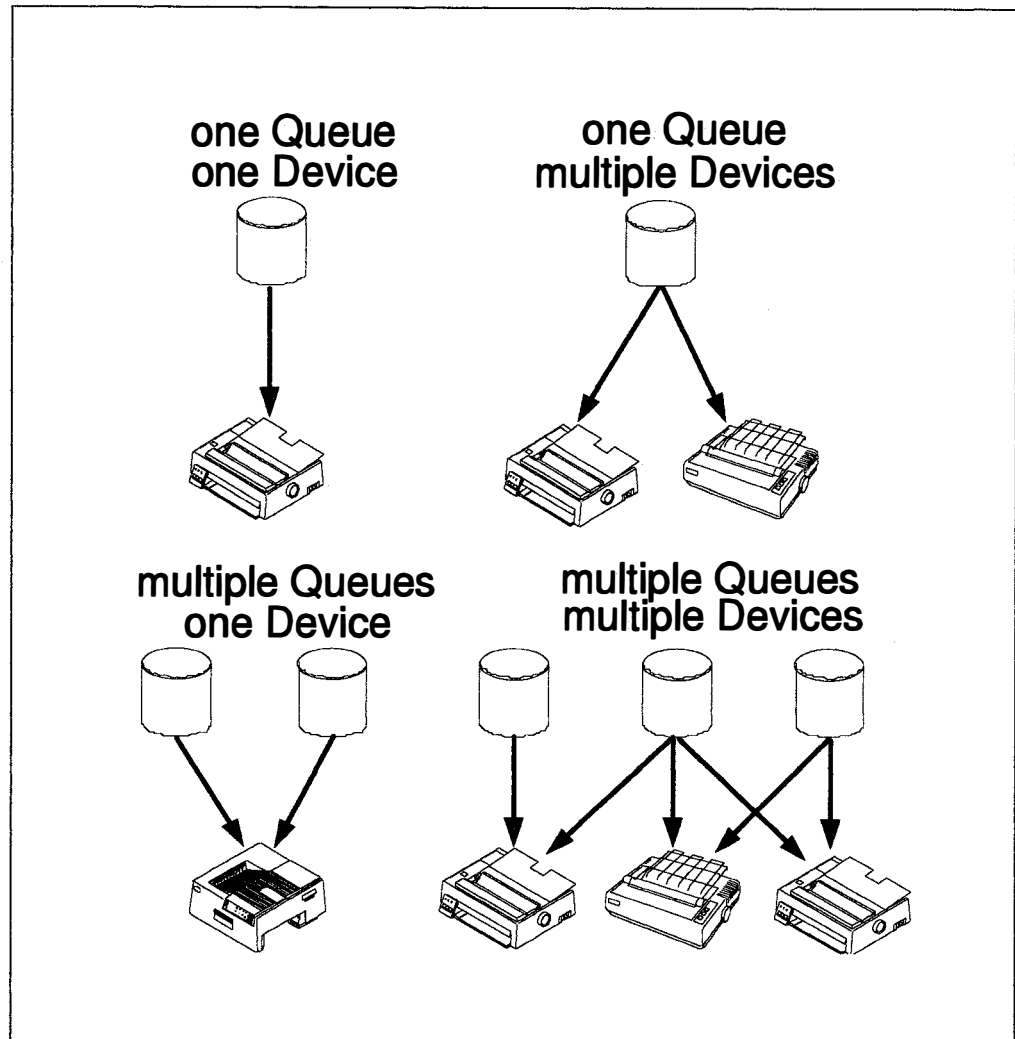


Figure 18-1. Example Configurations of Queues, Devices, and Printer Objects

Multiple queues connected to a single device is termed *printer sharing*. The advantage of printer sharing is that two queues can have different configurations or be used for different purposes. For example:

- One queue could be used for small jobs that are needed quickly, and the other could be used for large jobs printed during times of low demand, such as overnight.
- One queue could be used for a normal form such as a letter, and the other could be for a special form and so, would be held. When the special form was loaded into the printer, that queue could be released and the normal form queue could be held.

A single queue connected to multiple devices is termed *printer pooling*. This allows print jobs to be shared among printers for *load balancing*. The spooler does this by printing a job on the next available printer. For example one printer usually reserved exclusively for special forms could be connected to a normal form queue during peak loads. The special form queue would be held temporarily.

Note: Previous versions of OS/2 allowed the configuration of a device without a queue. OS/2 2.0 does not allow this configuration as it can lead to configuration problems for the user.

The logical device has the following configuration parameters that are relevant to application programmers:

- Name and description
- Logical port (for example, LPT1)
- List of printer drivers.

For example, a logical device representing an HP LaserJet printer with an installed PostScript option could have both the HP LaserJet printer driver and the PostScript printer driver in its configuration.

Each printer driver in the list also could have printer properties associated with it. *Printer properties* describe the physical configuration of the printer, such as what size paper is installed.

The queue has the following configuration parameters that are relevant to application programmers:

- Name and description

It is the queue *description* not the queue *name* that is used for the printer object title; that is, the text displayed beneath a printer object icon.

- Printer driver
- Default job properties

Job properties describe the parameters that must be used for printing a specific job; an example of a job property is the orientation. The queue's job property defaults are used if none are supplied by the application.

- Queue driver
- Flags for **Printer-specific format** and **Print while spooling** settings
- Separator page
- Start and stop times.

The configuration data for printer objects is stored in the OS2SYS.INI file. The spooler provides a set of functions that can be used to query and set this data. See "Spooler Management and Configuration" on page 18-31 for further details.

Note: Older PM applications might still use the Profile functions (for example, PrfQueryProfileString). If so, these applications must be recoded with the new functions to avoid any dependencies on where and how the system stores configuration data.

Printing Data Flow

The data flow between your application, the printer driver, the spooler, and the kernel device driver is shown in Figure 18-2 on page 18-7. In addition to the data flow for the printer data stream, the data flow for screen output is shown.

There are 3 routes that the application's printer data can follow:

- **Base Printing**

Base printing is provided primarily for non-PM programs writing complete printer data streams, including all printer control codes, directly to a printer port. It also provides compatibility for applications that run under DOS or Microsoft® Windows™. See "Submitting a Non-Presentation Manager (Base) Print Job" on page 18-9 for details about base printing.

- **Queued PM Printing**

Queued printing is recommended for all PM programs. It provides the most flexibility, both for the application and the user.

Print jobs created by PM applications are queued on a spooler queue. The spool files are processed and finally sent to the printer asynchronously from the application. See "Submitting a Queued Presentation Manager Print Job" on page 18-9 for more information.

Note: If the spooler is disabled, queued print jobs perform as though they were submitted for PM direct printing.

- **Direct PM Printing**

Conceptually, *direct printing* is the same as queued printing for the application interface, but the spooler is bypassed. Therefore, the data is sent directly to the printer; the application has to wait until the printing is finished. Following are some of the reasons for PM applications to avoid direct printing:

- The print job can interfere with other users whose jobs are destined for this printer. Print output could be mixed with the other print jobs
- The spooler allows only one job at a time to print to a particular port to avoid mixing of print job output. If your application tries to print directly to the same port while another job is printing, your user must wait for the current job to complete before the user's job can begin.
- The printing application loses some of its multitasking advantages because one job must complete before a second job is submitted.
- The printer device driver does not protect a print job from job property and printer property mismatches.
- Pictures printed directly can be different from those printed through a spooler. The default setting for direct printing is to print pictures their actual size, whereas the default for queue printing is *scaled to fit* for the output area.

Direct printing is recommended only for specialized applications that use dedicated hardcopy devices. Print jobs that have a certain degree of security associated with them, such as a corporate payroll or confidential documents, might best be handled with direct printing because files in the spooler can be copied.

Jobs that are very large, that is, over 10MB, that you do not want copied in the spool file, also might best be printed directly. See "Submitting a Direct Presentation Manager Print Job" on page 18-30 for details about direct printing.

Submitting a Non-Presentation Manager (Base) Print Job

A program can create a complete printer data stream including all printer control codes (called a *raw data stream*) by using `DosOpen`, `DosWrite`, and `DosClose`. This is called *base printing* and it is used by:

- OS/2 and DOS **Print** command
(for example, `PRINT CONFIG.SYS /D:LPT2`)
- OS/2 and DOS **Copy** command
(for example `COPY CONFIG.SYS LPT1:`)
- OS/2 and DOS **Redirected Output**
(for example `DIR > LPT1:` or `TYPE C:\CONFIG.SYS > LPT1:`)
(full screen hardcopy)
- All DOS applications
- All *family* applications (applications running under DOS and OS/2)
- All Microsoft Windows applications.

Non-PM applications developed under OS/2 2.0 that must generate graphical data must be able to generate all the necessary *Escape Codes*, or printer-specific control sequences, and will be coded differently, depending on the printer family; for example, Epson[®], HP[®] LaserJet[®], LaserPrinter[®], Proprinter[®], or PostScript[®].

Non-PM applications that successfully drove the printer in the environment for which they were designed—DOS for example—will continue to do so successfully under this operating system.

DOS applications that directly access output hardware registers could have difficulty printing under the operating system; if so, these applications might need to be rewritten.

Submitting a Queued Presentation Manager Print Job

There are several stages involved in using the PM programming interface to print your application data. From a user interface point of view, the following 4 dialogs must be provided. The exact order in which these dialogs are used will govern the order of program execution.

- **Page setup dialog**

The *page setup dialog* enables the user to specify the form name or size required. Options on this dialog can include margins on the page, header and footer strings, and whether duplex formatting is required.

The actual contents of the dialog depend on the type of application. The important factor is that this page specify formatting options that also are used to display to a screen.

For details about supporting a page setup dialog, see “Page Setup Dialog” on page 18-10.

- **Printer setup dialog**

The *printer setup dialog* displays a list of queues available to the user. A **Job properties** push button on this dialog enables the user to query and modify the job properties for this job. You can find more information about job properties in “Job Properties Considerations” on page 18-14.

For details about supporting a printer setup dialog, see “Printer Setup Dialog” on page 18-13.

- **Font dialog**

Most PM applications must enable the user to specify the font (or fonts) required. Once the user has chosen a printer, an option on the fonts dialog enables the user to choose from device fonts as well as system fonts.

For details about supporting a font dialog, see “Font Dialog and Device Font Considerations” on page 18-17.

- **Print dialog**

The application should have a Print menu item on the File menu to invoke an application-specific print dialog. It is recommended that Shift + Print Screen be used as an accelerator for printing the client area. The Print-Screen key, unshifted, is used to capture and print a window or the whole screen.

For details about supporting a print dialog, see “Print Dialog and Print Processing” on page 18-18.

Page Setup Dialog

The page setup dialog is concerned with formatting options for the document. The user must be able to specify the form name, margins, and other application-specific formatting options such as page duplexing; that is, different formats for left and right pages in a multi-page document. See Figure 18-3 for an example of a page setup dialog.

Note: The application is responsible for storing the user-defined margins for the form. The application must not allow the user to specify margins smaller than those returned by the printer driver.

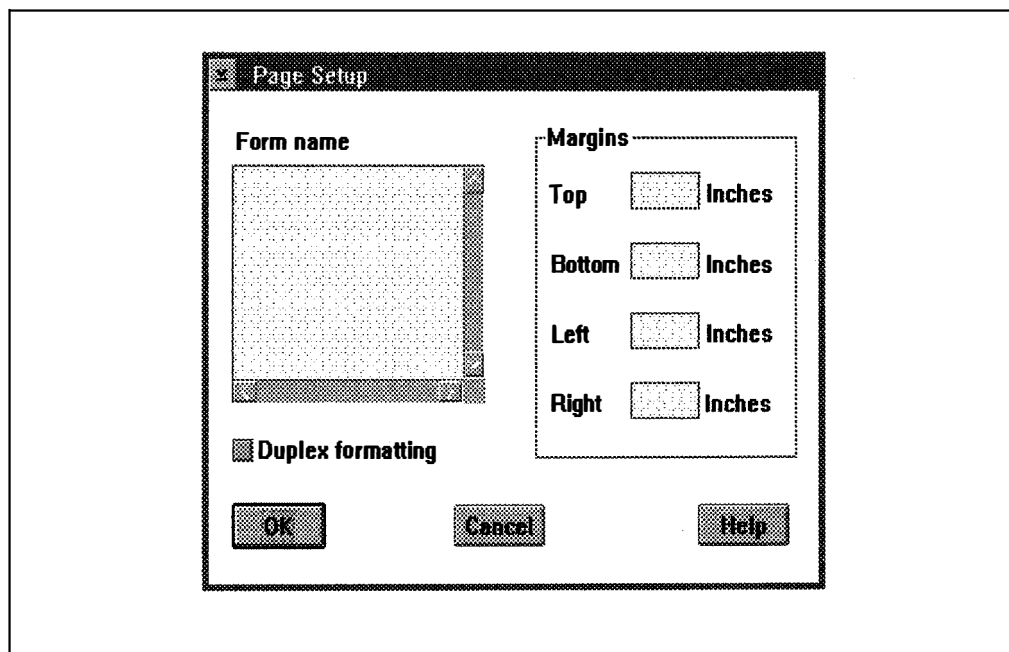


Figure 18-3. Application Page Setup Dialog

Forms Selection

If the user has not chosen a specific printer, the application can supply a list of standard form sizes; for example, Letter, Legal, Ledger, A4, and A3. The application also must consider, at a minimum, a 0.4 inch (10mm) margin on the form to be within the hardware clip limits of most printers. See Table 18-1 for the common form sizes.

<i>Table 18-1. Common Form Sizes</i>		
Form Name	Size in inches	Size in mm
Letter	8.5 x 11	216 x 279
Legal	8.5 x 14	216 x 356
Ledger	11 x 17	279 x 432
A4	8.3 x 11.7	210 x 297
A3	11.7 x 16.5	297 x 420

When a printer is chosen, it must be queried for the forms it supports and the hardware clip margins; use `DevQueryHardcopyCaps`. First however, a device context must be created. It is recommended that a `OD_INFO` context be used, because `OD_QUEUED` can result in the creation of a print job. See Figure 18-4 on page 18-12 for a sample code fragment.

```

PPRDINFO3    pprd3Device;           /* From SplQueryDevice */
PPRQINFO3    pprq3Queue;           /* From SplQueryQueue */
PSZ          pszTmp;               /* Temporary pointer */
HDC          hdc;                  /* Device context handle */
DEVOPENSTRUC dopData;              /* DEVOPEN structure */
LONG         clForms;              /* Number of forms */
PHCINFO      pchinfo;              /* The device */
ULONG        ulrc=DEV_OK;          /* Forms information */
                                           /* structure */
                                           /* Return code */

/* Fill in data for devopendata for OD_INFO */
dopData.pszLogAddress = pprd3Device->pszLogAddr;
pszTmp = strchr(pprq3Queue->pszDriverName, '.');
if (pszTmp)
    *pszTmp = '\\0';
dopData.pszDriverName = pprq3Queue->pszDriverName;
dopData.pdriv = pprq3Queue->pDriverData;

/* Open the information device context */

hdc = DevOpenDC ( (HAB)0,
    OD_INFO,           /* Type */
    "",               /* Default token */
    3L,               /* Count */
    &dopData,          /* Pointer to data */
    (HDC)0);          /* Comp dc */

/* Query number of forms available on the device */
clForms = DevQueryHardcopyCaps(hdc,
    0L,               /* Start at beginning of list */
    0L,               /* Get number of forms */
    NULL);

/* Allocate memory block for forms */
if (DosAllocMem((PPVOID(&pchinfo), clForms*sizeof(HCINFO), fALLOC))
{
    DevCloseDC(hdc);
    return(DEV_ERROR);
}

/* Query forms data */
ulrc = DevQueryHardcopyCaps(hdc,
    0L,               /* Start with first form */
    clForms,          /* Query all forms */
    pchinfo);         /* Structure to hold returned */
                                           /* data */

/* Close the information device context */
DevCloseDC (hdc);    /* Close the information */
                                           /* device context */

/* Now use forms information in pchinfo */

```

Figure 18-4. Querying the Printer Using DevQueryHardcopyCaps

DevQueryHardcopyCaps returns one HCINFO structure for each form. The contents of the structure are:

- ASCIIZ name of the form (for example, Letter)
- Width and height (in millimeters) of the paper
- Clipping limits (in millimeters) of the paper

- Number of pels between clipping limits
- Whether this form is the one installed currently on the device, or whether it is selectable from another paper bin.

Then the list of forms can be displayed to the user. The form preselected in the list should be one of the forms marked with the HCINFO structure flag HCAPS_CURRENT.

It is recommended that an application indicate to the user which forms are currently installed in the printer. This is done by including the HCINFO structure flag HCAPS_SELECTABLE. Then users can decide whether they want a quick print on a form available from the printer or to install a different paper tray in the printer.

Printer Setup Dialog

In a printer setup dialog, an application should offer a list of printer objects that are available to the user and enable the user to select one. (The list of printer objects actually is a list of queues.) If none are available, an appropriate message must be displayed. An application must query the list of available printer objects each time the printer setup dialog is displayed because the user might have created or modified the printer configuration while the application was executing. Figure 18-5 is an example of a printer setup dialog.

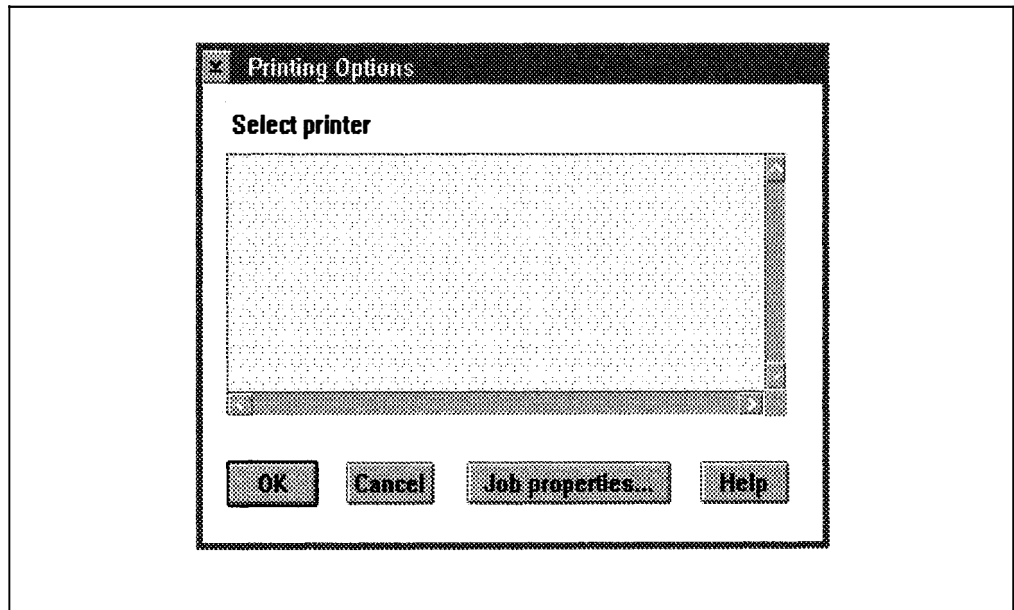


Figure 18-5. Application Printer Setup Dialog

Note: If the document was formatted for a particular device and the user selects a different printer, the application must ask the user's permission before reformatting the document for the new printer.

Use `SplEnumQueue` to query the list of printers (printer objects); *printer objects* essentially are spool queues. `SplEnumQueue` returns both a list of queues and information about each queue on the local workstation in an array of `PRQINFO3` structures. It also returns information about local workstation queues that reference network print queues. See "Network Printing Considerations" on page 18-26.

Because the number of queues might vary for each use of your application, it is essential to allocate sufficient storage to hold the data returned by `SplEnumQueue`. Usually the application issues the query twice: the first time, the application

determines the necessary size of the information buffer; after allocating a memory block, the second query actually retrieves the information.

The `SplEnumQueue` parameter *pcTotal* contains the number of queues available on the local system. The application should display an appropriate message box if the value is 0.

The queue description (returned in the structure `PRQINFO3` field name **pszComment**) is the printer object title. This is much more familiar to the user than the queue name, which is displayed only on the view settings page of a print object. Therefore, the printer setup dialog should show the queue descriptions instead of the queue names.

`SplEnumQueue` returns information about the queue that might influence the user's or application's decision to print to this queue; for example, the queue priority, or the number of jobs already in the queue. `SplEnumQueue` also returns the default job properties for the queue. This data can be used by the application for the **Job properties** push button on the printer setup dialog.

Less sophisticated applications might decide to dispense with the printer setup dialog and just print to the default queue. The default queue can be queried using `PrfQueryProfileString`, with an application name of `PM_SPOOLER` and a keyname of `QUEUE`. The spooler function `SplQueryQueue` then can be called to retrieve the default job properties.

Job Properties Considerations

Job properties are options on a per-job basis; for example, orientation, resolution, and form selection. The job properties dialog is displayed by the printer driver. Each driver has a different dialog, depending on the capabilities of the printer. The job properties are held in a printer driver-specific format in the **abGeneralData** field of the `DRIVDATA` structure.

Note: Job properties from one printer driver should not be given to another printer driver; the job properties probably will not be understood and the printer driver will either return an error or substitute some default job properties. Then the user will see a change in job properties stored previously.

The user should be given the opportunity to change the job properties before the job is printed. This can be achieved by supplying a **Job properties** push button on the printer setup dialog.

Changing job properties requires two steps:

- Retrieving job properties.
- Displaying the job properties dialog.

Retrieving Job Properties

The application can retrieve job properties from the following:

- Previously saved job properties with the document
- Current application session job properties
- Default job properties from the queue (from **pDriverData** in the `PRQINFO3` data structure)
- Printer driver device defaults (from `DevPostDeviceModes` using the `DPDM_QUERYJOBPROP` flag).

If no job properties were saved with the document, then there may be job properties that are being used by another document that also is to be printed to the same queue.

If job properties still cannot be found, then the default job properties stored with the queue can be used. It may be that the user has not set up any default job properties for the queue. Last, query the printer driver for its device defaults using `DevPostDeviceModes` with the `DPDM_QUERYJOBPROP` flag.

Displaying Job Properties Dialog

The sample code in Figure 18-6 on page 18-16 shows how to display the job properties dialog. All the parameters required are available from the `PRQINFO3` structure returned by `SplEnumQueue` or `SplQueryQueue`.

In the case of printer pooling, the `pszPrinters` field of the `PRQINFO3` structure contains a list of device names, separated by commas. It is sufficient to choose the first printer in the list because the print object ensures that the configuration of each spooled printer is the same.

Note: It is possible that the printer driver now uses a larger buffer for the job properties than the application is expecting. This occurs when a new version of a printer driver that supports some additional features is installed.

In this case, the application must discard the existing document job properties, after a confirmation from the user, and query the printer driver for its device defaults, using the `DPDM_QUERYJOBPROPS` parameter to `DevPostDeviceModes`.

```

#define INCL_DEV
#define INCL_DOS
#include <os2.h>
#include <memory.h>

{
    ULONG          ulrc=FALSE;
    HAB            hab;
    PPRQINF03      pprq3Queue; /* From SplEnumQueue or SplQueryQueue */
    PSZ            pszDriverName, pszDeviceName, pszTmp;
    HDC            hdc=NULL;
    LONG           cbBuf;

    /* Use the first device name in the PRQINF03 structure */
    pszTmp = strchr(pprq3Queue->pszPrinters, ',');
    if (pszTmp)
        *pszTmp = '\\0';

    /* Use just the driver name from the driver.device string */
    pszDeviceName = strchr(pprq3Queue->pszDriverName, '.');
    if (pszDeviceName)
    {
        *pszDeviceName = '\\0';
        pszDeviceName++;
    }

    /* Check size of buffer required for job properties */
    cbBuf = DevPostDeviceModes( hab,
                                (PDRIVDATA)NULL,
                                pprq3Queue->pszDriverName,
                                pszDeviceName,
                                pprq3Queue->pszPrinters,
                                DPDM_POSTJOBPROP
                                );

    /* Return error to caller */
    if (cbBuf<=0)
        return(cbBuf);

    /* Return BUFFER TOO SMALL error to caller */
    if (cbBuf > pprq3Queue->pDriverData->cb)
        return(DPDM_ERROR);

    /* Display job properties dialog & get updated job properties from driver */
    ulrc = DevPostDeviceModes( hab,
                                pprq3Queue->pDriverData,
                                pprq3Queue->pszDriverName,
                                pszDeviceName,
                                pprq3Queue->pszPrinters,
                                DPDM_POSTJOBPROP
                                );

    return(ulrc);
}

```

Figure 18-6. Displaying the Job Properties Dialog

Font Dialog and Device Font Considerations

There are two types of fonts:

- *System fonts*—can be used for both displays and printers, and therefore, are generally more flexible.
- *Device fonts*—specific to a particular device. Device fonts for printers can be built-in or be installed with font cartridges by a user. Device fonts also can be available on diskette (also termed *soft fonts*) and can be downloaded to the printer as required by the printer driver.

The advantage of device fonts is that, usually, they are printed in the highest resolution of the device and are faster than system fonts.

Note: Some printer drivers, in particular, the HP LaserJet (LASERJET) and the LaserPrinter (IBM4019) printer drivers provide an intermediate solution. An option on the job properties dialog enables system fonts to be downloaded to the printer as soft fonts.

As stated earlier, an application must provide the user with the ability to choose fonts and, in particular, to choose a device font over a system font to achieve better performance. A standard font dialog box should be used (see WinFontDlg).

When an application uses device fonts, and the output mixes text and graphics pictures, the device fonts are clipped per character. System fonts give more precise clipping to the pel. Figure 18-7 illustrates the results from clipping two lines of text: one generated in a system font; the other, in a device font.

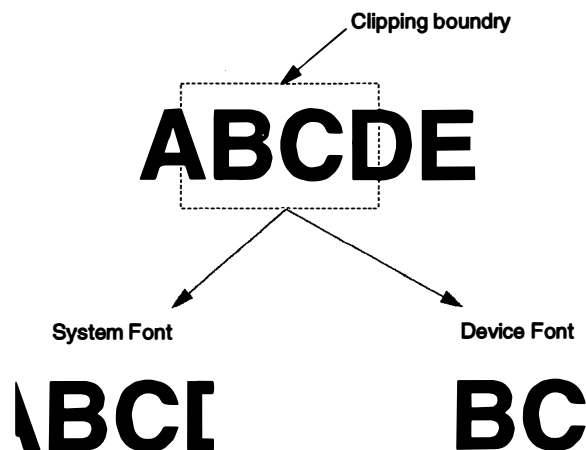


Figure 18-7. Character Clipping Results for Device and System Fonts

The available device fonts can be queried, using `GpiQueryFonts`, in either an `OD_INFO` or an `OD_QUEUED` device context. Device fonts are returned with negative *IMatch* numbers. Then the appropriate logical font can be used after issuing `GpiCreateLogFont`.

If, during the printing process, a logical font is not created, the printer driver uses its default font for the printer, which usually is 12-point Courier.

There are two design choices for device fonts:

- Use system fonts for the display. When printing, query the printer driver and attempt to choose a device font that closely matches the system font. If no match

is found, then print using the system font. A device's font metrics and character spacing might not match exactly, so the printed output might not be exactly the same as the display.

- Use a printer font if the user selects one. Determine the metrics and find a system font that shares the same characteristics (for example, a style such as Courier or Serifed). The character string to be displayed is sent to the printer driver and the inter-character spacing is returned, using `GpiQueryCharStringPos`. The individual character positions are used when the string is sent to the display, using `GpiCharStringPos` with the `fOptions` parameter of `CHS_VECTOR`. While the display output can take longer to display to the screen, true WYSIWYG (*What You See Is What You Get*) is achievable.

Print Dialog and Print Processing

The print menu must display a dialog to enable the user to specify the number of copies, start page, end page, or any other application-specific print options. A confirm push button (for example, **Print**) initiates the print. Figure 18-8 is an example of a print dialog.

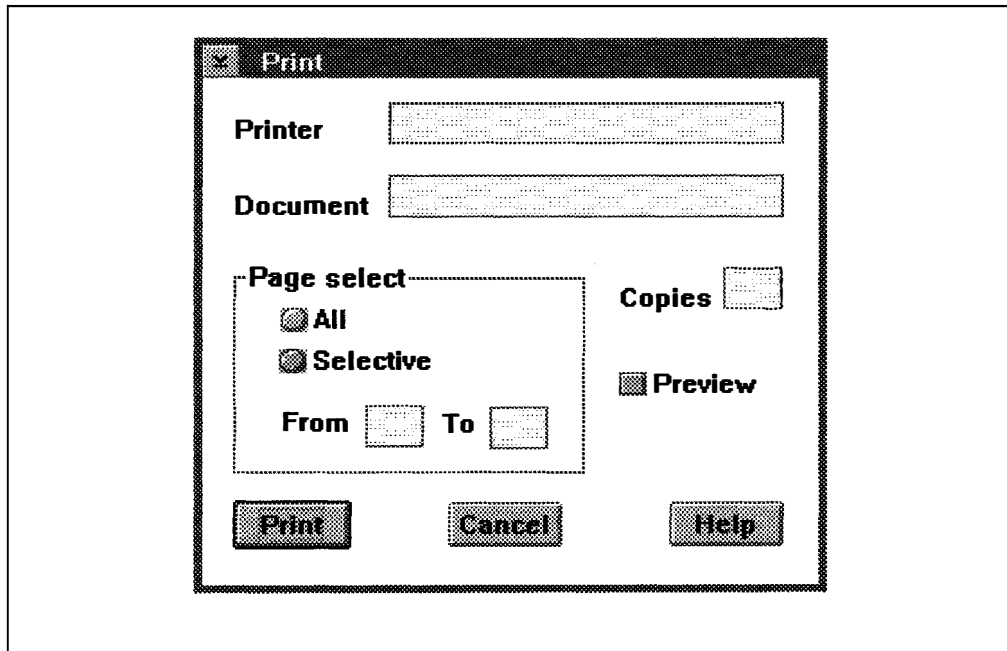


Figure 18-8. Application Printer Setup Dialog

The **Printer** entry field is the read-only name of the printer object (queue) that the user chose from the Printer Setup dialog. The **Preview** checkbox is used to preview the printed output on the screen. This can be achieved by opening an `OD_MEMORY` device context to the printer driver and drawing the picture into the bit map. A Bit Blt to the screen shows the resultant page. System fonts should be used instead of device fonts.

Once the user has initiated the print, a PM application must execute the following steps.

1. Create a new thread.

Then, on this new thread:

2. Open a device context.
3. Associate a presentation space.

4. Start the print job.
5. Query the device capabilities.
6. Format the page.
7. Start the next page.

Repeat steps 6 through 7 for each page.

8. End the print job.

For another print job, repeat from step 4.

9. Disassociate the presentation space.
10. Close the device context.

The following sections describe each of the above steps in turn.

Creating a New Thread

Once the print process has been confirmed by the user, the application must use a separate thread to perform the actual printing. This maintains user responsiveness and avoids displaying the hourglass pointer. While the printing is in progress, the application should dim certain menu options, such as drawing. Other options, such as page down, zoom, or help still must be available.

Opening a Queued Device Context

Before you can send data to a printer using device functions, you must open a device context with `DevOpenDC`. The parameters of `DevOpenDC` are influenced by the queue and job properties the user has chosen previously.

`DevOpenDC` accepts the following parameters as input:

hAb	The anchor-block handle from a <code>WinInitialize</code> call.
IType	The type of device context. This always is <code>OD_QUEUED</code> to specify a queued device context.
pszToken	The device-information token. Always use an asterisk (*) for this parameter to force the system to get device information from the <i>pdopData</i> parameter.
ICount	The number of data elements supplied in the <i>pdopData</i> parameter. The minimum number of parameters for a queued device context is 4.
pdopData	The device-context data area. This is a pointer to a structure of the type <code>DEVOPENSTRUC</code> . The elements of this structure are described below.
hdcComp	The compatible-device-context handle. This must be <code>NULL</code> for a device context of type <code>OD_QUEUED</code> .

`DevOpenDC` returns a device-context handle of type `HDC`. The handle is used in other functions beginning with the `Dev` prefix and for `GpiCreatePS` and `GpiAssociate`.

The `DEVOPENSTRUC` structure contains all the data needed to define a device context.

The individual elements of the DEVOPENSTRUC structure are described below. At least the first 4 structure members must be provided for queued device contexts.

pszLogAddress	Name of the queue. It is the pszName field of the PRQINFO3 structure. Note: For device contexts of type OD_INFO, pszLogAddress is the port name. This can be retrieved by calling SplQueryDevice using the pszPrinters device name. The pszLogAddr field in the PRDINFO3 structure is the port name.
pszDriverName	Character string identifying the printer driver, for example, LASERJET. The pszDriverName field of the PRQINFO3 structure, associated with the required print queue, gives the driver and device name, separated by a period, for example LASERJET.HP LaserJet IIID. The pszDriverName field can contain only the name up to the period, for example LASERJET.
pdriv	This is a pointer to the job properties data returned by the printer driver from DevPostDeviceModes or the default job properties from pDriverData in the PRQINFO3 structure. The DRIVDATA structure is described in the <i>Presentation Manager Programming Reference, Volume III</i>. Note: The DRIVDATA structure contains the particular device name to be used. Therefore, it is a programming error to set this parameter to NULL.
pszDataType	It is recommended that PM_Q_STD always be used for the data type.
pszComment	Optional character string that the printer object displays to the user in a job settings notebook. It is recommended that the application include its own name in this comment string. Note: The job title text is derived from the document name (see "Starting a Print Job" on page 18-22).
pszQueueProcName	Queue processor name (optional). The queue processor (also termed <i>queue driver</i>) name is available from the pszPrProc field of the PRQINFO3 structure. The default queue processor provided by the operating system is PMPRINT. The user also can install a queue processor (PMPLLOT) that is used to provide reverse clipping for vector devices such as plotters. For specialized applications, it is possible to use an alternative queue processor to the default specified for the queue. The list of installed queue processors is available from the OS2SYS.INI file using the application name PM_SPOOLER_QP.
pszQueueProcParams	Queue processor parameters (optional). They can include information such as the number of copies you want to print and the size of the output area on the printed page. See "PMPRINT/PMPLLOT Queue Processor Parameters" on page 18-27 for details.
pszSpoolerParams	Spooler parameters (optional) are separated by spaces. They are used for scheduling print jobs and are as follows: • The form names that identify the paper to be used, for

example, *FORM=A4,A5,ENV*. The form names are optional; but if they are provided, the spooler is able to hold off printing the jobs until the required form is installed in the printer. If the form name is not provided, the spooler attempts to print the job. The printer driver recognizes that there is a forms problem and displays a FORMS MISMATCH message box.

- Priority of the print job, for example, *PTY=60*. The priority is specified as an integer in the range 1 through 99; 99 is the highest. The default priority value is 50. The application can use the spooler priority parameter to prioritize its own jobs. However, it is not good practice for an application always to use priority 99 in an attempt to get its jobs printed first.

pszNetworkParams Optional parameter that can be used to specify network options; for example, *USER=JOESMITH*.

Associating a Presentation Space

In order to print, a device context must be associated with a GPI presentation space (PS).

In most circumstances, a presentation space already exists for the screen. In such cases, the PS can be disassociated with the window device context and associated with the printer device context by using `GpiAssociate`. See Figure 18-9.

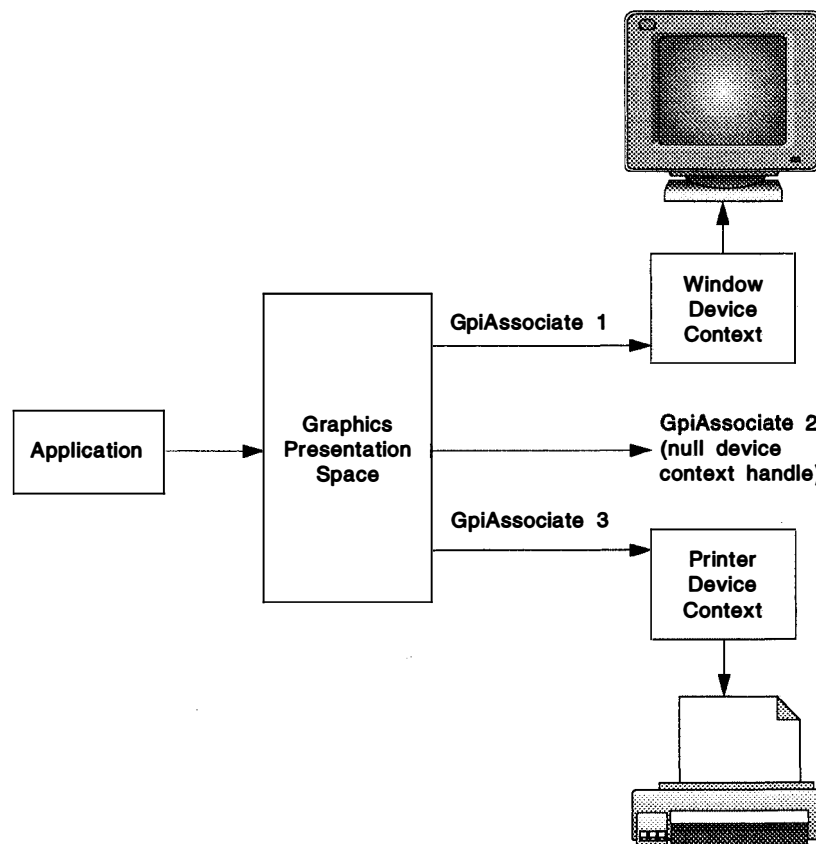


Figure 18-9. Reassociating Presentation Space with Device Contexts

If a presentation space does not exist or a different one is to be used, the application must issue GpiCreatePS and use the GPI_ASSOC flag to associate the printer device context.

Device-Independence Considerations

The operating system supports true WYSIWYG. The same GPI functions the application used to create the picture or document on the display screen can be used to create the output on a printer. This is the major advantage of device independence.: An application must be designed around the options provided by the PM programming interface to ensure device independence, as a display (VGA) typically is 96 dpi, while a printer typically is 300 dpi.

Following are some application design considerations:

- Use DevQueryHardcopyCaps to determine the horizontal and vertical width of the form and the maximum printable area.
- Create a presentation space with a presentation page size of 0 to ensure that the maximum window size and maximum page size are used.
- Use device-independent coordinates, such as LO_ENGLISH, or TWIPS. If you program in pels, transferring your output from screen to hardcopy will compress the image. See Figure 18-10.

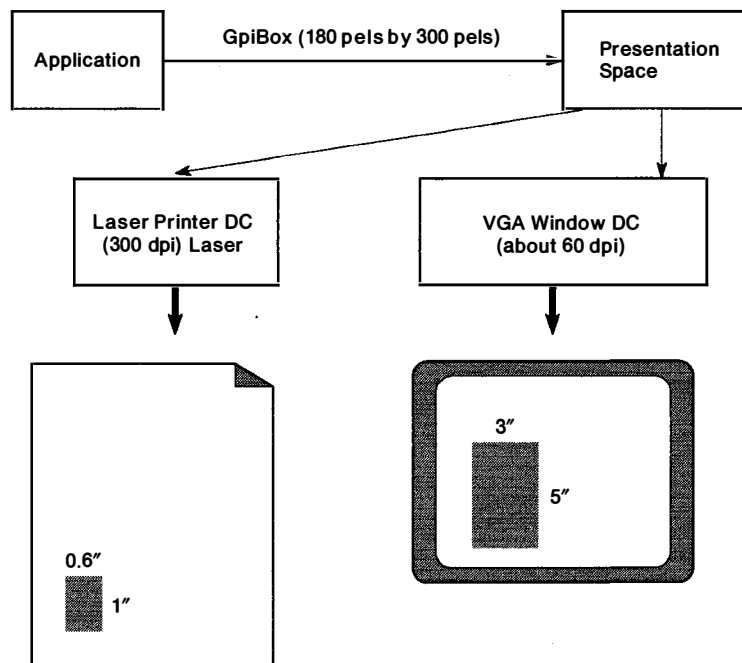


Figure 18-10. An Effect of Programming in Pels. Reassociating a presentation space defined in pels or other device dependent units can affect the dimension of the picture. If the pels are not square on the display device, the shape and scale (aspect ratio) will be different as well.

Starting a Print Job

The application signals the beginning of a print job by using DevEscape with the escape parameter DEVESC_STARTDOC. The application must specify a document name; usually this is related to the name of file being printed. Any GPI calls made before this DevEscape(DEVESC_STARTDOC) are ignored.

Querying the Device Capabilities

DevQueryCaps can be used to provide over 40 pieces of information about a specific device; for example, default character box size, horizontal and vertical resolution, or number of device-specific fonts.

Knowing that the device has only 1-bit color support (monochrome) might influence the application to use line style and patterns instead of colors for output.

Formatting the Page

Before print processing actually starts, it is recommended that an application's modal dialog be displayed, showing page progress and enabling the user to cancel the print job. If the user presses **Cancel** on this dialog, the application must issue DevEscape with the parameter *DEVESC_ABORTDOC*. This allows the spooler and printer driver to clean up. A job is not created in the queue.

Usually the output on a page is generated by a series of GPI calls from the application. By calculating the size of character strings or graphics written to the presentation space, the application must be able to decide when to move to the next page in a multi-page document.

Starting the Next Page

When the application has finished processing a page, it must issue DevEscape with the *DEVESC_NEWFRAME* parameter. The progress dialog must be updated. The DevEscape *DEVESC_NEWFRAME* can be used to generate blank pages. The printer driver always issues a page eject whenever this DevEscape is received.

Issuing the *DEVESC_NEWFRAME* escape does not reset any attributes or fonts. However, the bounds and any clipping regions are reset.

Ending the Print Job

The end of a print document is signalled by the application's issuing DevEscape with the *DEVESC_ENDDOC* parameter. The spooler returns a job identifier that can be used for job manipulation.

Figure 18-11 on page 18-24 shows the processing carried out by a printer driver as it receives device escapes and graphics from the GPI.

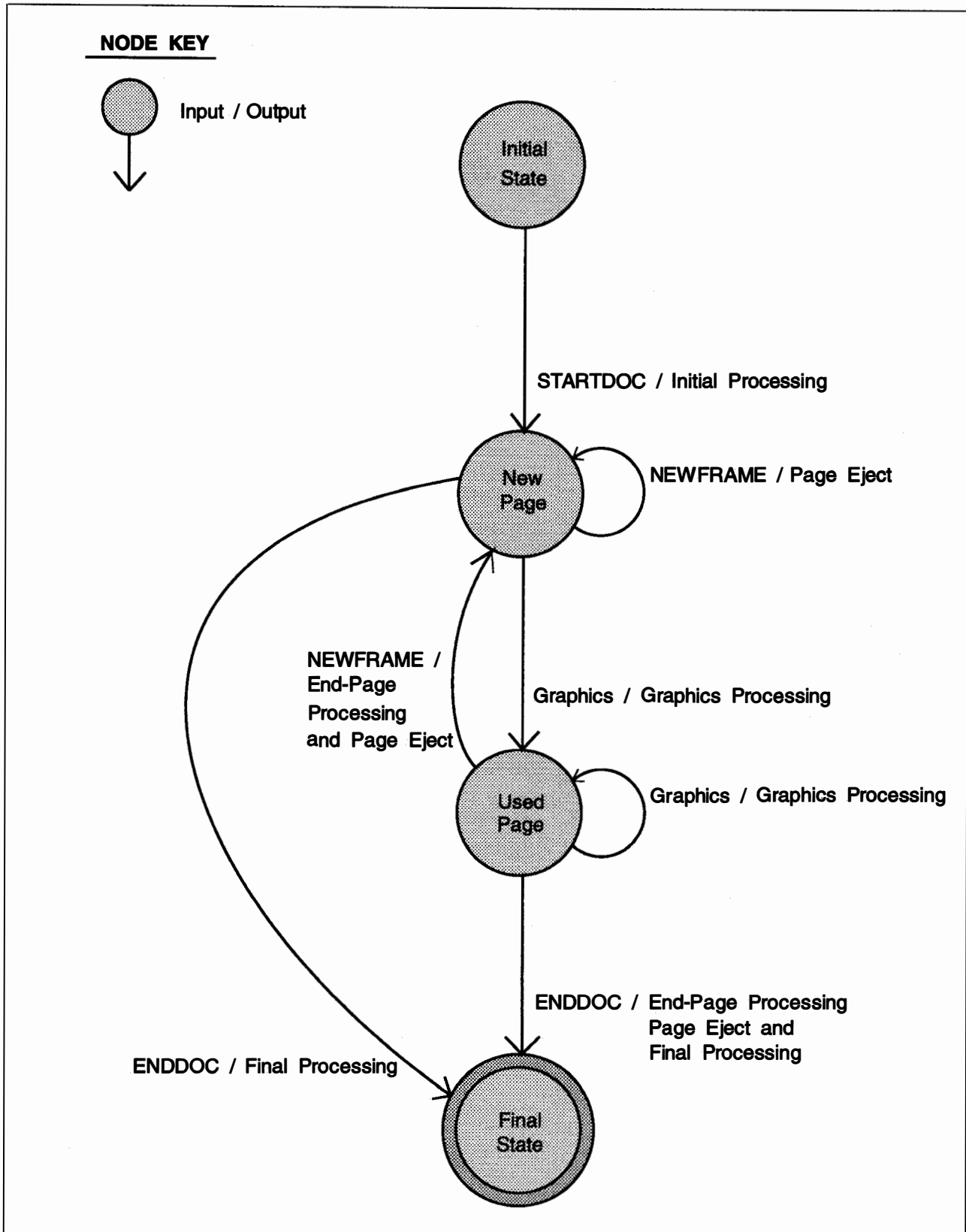


Figure 18-11. Flow for Device Escapes

Note: Each device-escape pair of *DEVESC_STARTDOC* and *DEVESC_ENDDOC* creates one print job in a spooler queue. If the application needs to create multiple print jobs, use the following sequence:

```

DevOpenDC
GpiCreatePS
DevEscape(DEVEESC_STARTDOC)
:
DevEscape(DEVEESC_ENDDOC)
DevEscape(DEVEESC_STARTDOC)
:
DevEscape(DEVEESC_ENDDOC)
DevCloseDC

```

Disassociating the Presentation Space

When the print job is complete, the presentation space can be disassociated with the printer device context using `GpiAssociate` with a `NULL` parameter. Then the presentation space can be reassociated with the display device context.

Closing the Device Context

A device context is closed using `DevCloseDC`. The print thread must signal print termination to the main processing loop. The main loop can terminate the print thread and dismiss the Page Progress message box.

An application can display a message box to the user indicating a successful print and show the job ID returned by the spooler.

Print-to-File Considerations

Print-to-file means that the print data destined for a printer is stored in a file instead. The advantage of doing this is that the file can be used later, possibly on a different machine or different environment, and the data printed without requiring the original application.

The system provides 3 ways to print to file:

- The user can specify **Print to file** on the **Output** settings page of a printer object. When a print is requested, the system displays a dialog in which the user can enter a file name.

This method is preferred because the application does not have to be aware; the user has full control of the system.

- The application can specify `OD_DIRECT` on a `DevOpenDC` call. The **pszLogAddress** field of the `DEVOPENSTRUC` structure specifies the file name as shown in the following example:

```

C:\TMP\MYPRINT.DAT
\\SERVER\DISK1\MYPRINT.DAT    -- UNC file name
\\PIPE\PRTPPIPE1             -- A pipe
LPT1                         -- A port device

```

- The application can create the printer-specific format data and write it to a file using `DosOpen`.

Network Printing Considerations

The architecture of PM requires a printer driver during the print job creation process to deal with queries, such as `DevQueryHardcopyCaps`, and to provide the job properties dialog by way of `DevPostDeviceModes`. To do any printing across a network, a locally installed printer driver is required. In some cases, such as when the network server does not run PM, this is an advantage, in that all the conversion to printer-specific commands can be done on the requestor.

The printer object in the workplace provides seamless access to network printers; the user need only install the appropriate printer driver.

The application programmer also is helped. A network printer that is accessed by the user has a hidden shadow on the requestor. The job properties for the shadow printer object can be altered and are stored on the requestor. This enables the user to configure one or more variations on the original configuration without requiring intervention from a system administrator to create many network printer objects, each with a slightly different default job property configuration.

The shadow printer object is created on the requestor by creating a local queue and local device with no port connection. The application can enumerate these queues using `SplEnumQueue`, as before. The only difference is in the `PRQINFO6` structure; there are two extra fields not in the `PRQINFO3` structure. These two extra fields, named **`pszRemoteComputerName`** and **`pszRemoteQueueName`** contain the name of the remote server and the remote queue. The spooler uses these fields to automatically reroute print data, submitted by an application to the local shadow, to the appropriate network queue.

The printer object creates a local shadow only when the network print object has been used or modified, because there could be a large number of network printer objects, and the spooler does not know which one the user wants to use. The local shadow of a network printer object is created in the following cases:

- When a file is dragged and dropped on the network printer object
- When the network printer object is moved, copied, or shadowed outside its original server folder
- When the printer or job properties are modified.

Drag/Drop Protocol Considerations

When an application data file is dropped on a printer object, the application can receive a `DM_PRINTOBJECT` message if the application is running currently.

One `DM_PRINTOBJECT` parameter gives a `DRAGITEM` structure that describes the object that was dropped. The other parameter gives a `PRINTDEST` structure that contains all the parameters required to issue `DevPostDeviceModes` and `DevOpenDC`.

The `PD_JOB_PROPERTY` flag indicates to the application that the user has requested a job properties dialog before printing. If this flag is not set, the application must not display the job properties dialog; it must use the job properties passed in the `PRINTDEST` structure.

The rest of the printing process follows the procedure described in "Submitting a Queued Presentation Manager Print Job" on page 18-9.

PMPRINT/PMPLLOT Queue Processor Parameters

How the queue processor processes a print job is controlled by the optional queue-processor parameters that you can specify in the *pdopData* parameter of DevOpenDC. The PM programming interface adds these parameter values to the spool file.

The PMPRINT/PMPLLOT queue processor parameters enable an application to do the following:

- Specify the number of copies of a print job
- Restrict printing to a specific area on the page
- Specify which part of a picture is to be printed
- Specify color output (if your printer allows this)
- Specify the foreground and background colors in a monochrome print.

Note: The application need not use queue processor parameters if all that is required is a single copy of a picture, scaled to fit the page. These are the default settings of the queue-processor parameters.

The first parameter (*COP*) is used for all spool-file formats. The remaining parameters are valid for PM_Q_STD spool files only. Because PM_Q_STD data are used mainly for *graphic* data, these parameters are described in relation to the printing of picture files.

The PMPRINT/PMPLLOT queue-processor parameters are separated by spaces and are:

COP = *n*

The *COP* parameter specifies the number of copies of the spool file that you want printed. The value of *n* must be an integer in the range of 1 through 999.

The default is *COP* = 1.

ARE = C | *w,h,l,t*

The *ARE* parameter determines the size and position of the output area. This is the area of the physical page to which printing is restricted.

The default value of *ARE* = C means that the output area is the whole page. Note, however, that the printer cannot print outside its own device clip limits.

To size and position the output area at a specific point on the page, use *ARE* = *w,h,l,t*, where:

w, h are the width and height of the desired output area.

l, t are the offsets of the upper-left corner of the output area from the left (l) and from the top (t) of the maximum output area.

These four values must be given as percentages of the maximum output dimensions. The maximum output area is the area within the device clip limits.

FIT = S | *l,t*

The *FIT* parameter determines which part of the picture is to be printed. You can request the whole of the picture, scaled to fit the output area; or you can position the picture (actual size) anywhere within the output area. This could mean that the picture is clipped at the boundaries of the output area.

The default value of *FIT* = S causes the output to be scaled until the larger of the height or width just fits within the defined output area. The aspect ratio of the picture is maintained.

To print the picture in actual size, use $FIT=l,t$, where l,t are the coordinates of the point in the picture that you want positioned at the center of the output area: l is measured from the left edge of the picture; and t is measured from the top edge. The coordinates must be given as percentages of the actual dimensions of the picture.

XFM = 0 | 1

The *XFM* parameter enables you to override the picture-positioning and clipping instructions that are provided by the *ARE* and *FIT* parameters, including their defaults.

The default value of $XFM=1$ allows the appearance of the output to be determined by the settings of the *ARE* and *FIT* parameters.

A value of $XFM=0$ yields output as specified in the picture file. For example, applications that use many different forms can define different positions on each form for their output.

COL = M | C

The *COL* parameter enables you to specify color output if you have a color printer.

A value of $COL=M$ creates monochrome output (black foreground with no background color). This is supported by all devices.

A value of $COL=C$ creates color output. If you request color output on a monochrome device, the printer presentation driver tries to satisfy your request, which can cause problems because the only color available is black. For example, if the picture file specifies a red line on a blue background, both are drawn in black.

The default is $COL=M$ when you are addressing a monochrome printer and $COL=C$ when you are addressing a color printer.

MAP = N | A

The *MAP* parameter enables you to decide how the *neutral* colors (those that are not specified in the picture file) are printed.

The default value of $MAP=N$ yields a *normal* representation of the screen picture on a printed page, which means that the page background is white and the foreground is black.

A value of $MAP=A$ provides the reverse of the normal representation: the background is black and the foreground is white on the printed page..

Notes:

1. Pictures from a spool file can be printed differently than those printed when the spooler is not selected, because the default setting without the spooler is to print pictures their actual size, whereas the default for the queue-processor *FIT* parameter is "scaled to fit" the output area.
2. The queue-processor parameters must be separated by one or more blank spaces; for example: $COP=3\ ARE=C\ FIT=S$.

The parameters can be listed in any order. Default values are used for parameters that are omitted or entered incorrectly.
3. Bit-map or image data (data constructed by setting pels on or off) is not affected by the *ARE* and *FIT* parameters. Consequently, if your picture contains bit-map or image data, the printed version will be different from the screen version if you use these parameters.

Examples Using the ARE and FIT Parameters

The application uses the *ARE* and *FIT* parameters to determine which part of the picture is printed, and where on the physical page it is printed. Figure 18-12 provides practical examples of how to use these parameters. Example 1 shows the output when both the *ARE* and *FIT* parameters are set to their default values; the result is a print of the whole picture, scaled to fit the full page. Examples 2 and 3 show how to restrict the output area to a portion of the page; because the *FIT* parameter is left at its default value, the whole picture is scaled to fit the restricted output area. Example 4 shows part of the picture printed actual size, using the whole page as the output area. Examples 5 and 6 show a part of the picture printed, actual size, in different positions on the page.

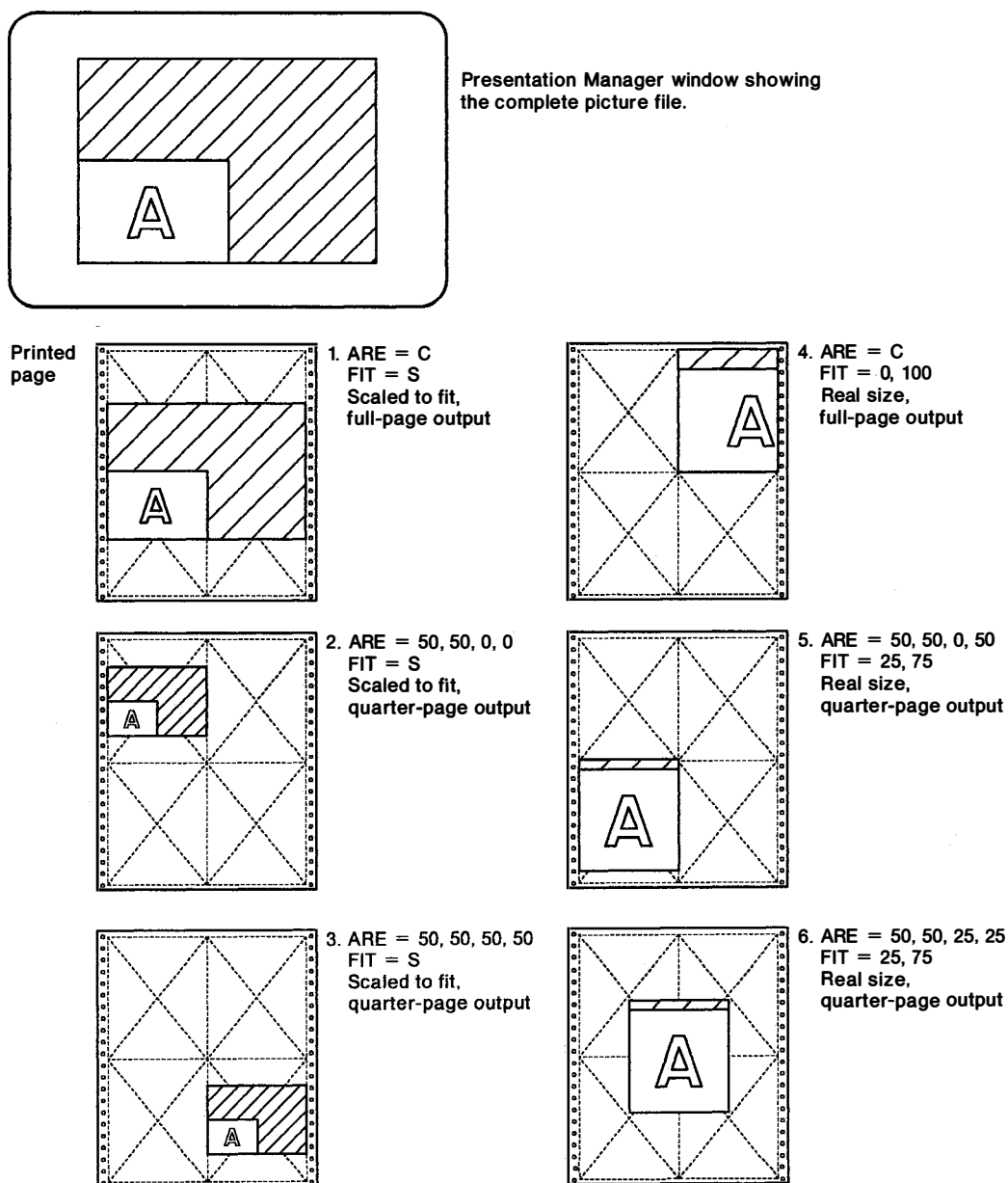


Figure 18-12. Examples Using the ARE and FIT Parameters

Submitting a Direct Presentation Manager Print Job

PM *direct printing* is done through a device context of type `OD_DIRECT`, as follows:

1. Output from the PM printer driver goes directly to the printer instead of to the spooler.
2. The application must specify `OD_DIRECT` in the `DevOpenDC` function.
3. The **pszLogAddress** field of the `DEVOPENSTRUC` structure is a port name (for example `LPT1`). The port name is held in the **pszLogAddr** field of the `PRDINFO3` structure returned by `SplEnumDevice` or `SplQueryDevice`.
4. All fields following the **pdrv** field in the `DEVOPENSTRUC` structure are ignored by the system.

The remaining steps in the printing process are the same as those for `OD_QUEUED` device contexts.

Submitting a Print Job Directly to the Spooler

The spooler provides functions that enable applications to submit print data directly to a spool queue. Normally, these functions are used by printer drivers to add a print job to a spool queue. However, this is not a recommended method unless the application is special-purpose.

The advantage of submitting print data directly to a spool queue is to bypass the GPI presentation layer. This can be useful, particularly, for sending print jobs to a network printer that is on a server that does not run the OS/2 operating system. However, there are certain requirements for direct spooling:

- Because the data bypasses the printer presentation driver, it must be in a format the printer can understand. Therefore, the application must be aware of and send the printer-specific commands.
- If the spooler is not active, the print jobs never will be printed.

The following steps are required to create a print job using the spooler directly:

1. Use `SplQmOpen` to open the PM programming interface. (Functionally, this is similar to using `DevOpenDC`, which is detailed on page 18-19.) You must specify a *queue name* of `PMQOPENDATA` for the logical address element, **pszLogAddress**. `PMQOPENDATA` is a data structure, identical to `DEVOPENSTRUC`, to be used with the `SplQmxxx` functions.
2. Use `SplQmStartDoc` to signal the start of your document. (This is same as using `DevEscape` with `DEVESC_STARTDOC` as used in PM printing.)
3. Use `SplQmWrite` to write data to the spool file.
4. Use `SplQmEndDoc` to signal the end of your document. (This is the same as `DevEscape` with `DEVESC_ENDDOC` as used in PM printing.)
5. Use `SplQmClose` to close the PM programming interface. (Functionally, this is similar to `DevCloseDC`.)

Spooler Management and Configuration

The spooler provides a set of functions that allows management of all aspects of the print subsystem. There are 5 categories:

- Job querying and manipulation (SplxxxJob)
- Queue creation, querying, and manipulation (SplxxxQueue)
- Device creation, querying, and manipulation (SplxxxDevice)
- Listing ports, printer drivers, queue processors (SplEnumyyy)
- Listing printers across a network (SplEnumPrinter).

The symbol xxx typically represents Add, Delete, Enum, Query, and Set. For jobs and queues, xxx also includes Hold and Release. For devices, xxx includes Control. The symbol yyy is either Port, PrinterDriver, or QueueProcessor.

See “Summary” on page 18-32 for a full list of functions. Also, refer to the *Presentation Manager Programming Reference, Volume I* for function details and sample code; see the *Presentation Manager Programming Reference, Volume III* for structure definitions.

It is anticipated that most applications will not use any spooler functions at all except SplEnumQueue, SplQueryQueue, and SplQueryDevice. However, there are some specialized applications that might use the spooler functions for the following:

- An administrative tool to manage queues and jobs
- An automatic printer object configurator.

A new printer object can be created by using SplCreateDevice and SplCreateQueue. The print subsystem recognizes that this has occurred and puts a printer object directly on the desktop.

When the spooler is disabled, the following spooler functions do not return any results:

SplControlDevice
SplCopyJob
SplDeleteJob
SplEnumJob
SplHoldQueue
SplQueryJob
SplReleaseQueue
SplReleaseJob
SplSetJob.

No function is available to enable or disable the spooler under application control; this is a user decision. It is recommended that the spooler always be enabled.

Print Job Management

Some sophisticated applications might need to manipulate jobs in the following ways after they have spooled into a queue:

- Modify the job parameters (for example, copies); use SplSetJob.
- Increase the priority of a particular job; use SplSetJob.
- Delete a job because a job based on newer data was created; use SplDeleteJob.

Summary

Table 18-2 summarizes the functions used to print jobs, manage jobs, and manipulate the spooler configuration.

<i>Table 18-2 (Page 1 of 2). Printing and Spooler Functions</i>	
Function Name	Description
DevCloseDC	Closes a device context.
DevEscape	Controls print job start, new page, and end.
DevOpenDC	Opens a device context.
DevPostDeviceModes	Displays the job properties dialog or retrieves device job property defaults.
DevQueryCaps	Retrieves information about the device's capabilities.
DevQueryHardcopyCaps	Retrieves information about forms.
SpiControlDevice	Cancels, holds, continues, or restarts a print device.
SpiCopyJob	Copies a print job (within the same print queue).
SpiCreateDevice	Creates a print device.
SpiCreateQueue	Creates a print queue.
SpiDeleteDevice	Deletes a print device.
SpiDeleteJob	Deletes a print job.
SpiDeleteQueue	Deletes a queue.
SpiEnumDevice	Lists print devices, optionally, with status information.
SpiEnumDriver	Lists printer drivers.
SpiEnumJob	Lists print jobs.
SpiEnumPort	Lists physical ports.
SpiEnumPrinter	Lists print queues and print devices, optionally, for the whole network.
SpiEnumQueue	Lists print queues.
SpiEnumQueueProcessor	Lists queue processors (queue drivers).
SpiHoldJob	Holds a job in a print queue.
SpiHoldQueue	Holds a print queue.
SpiPurgeQueue	Deletes all jobs in a print queue.
SpiQmAbort	Stops a print job and closes a spooler session.
SpiQmAbortDoc	Stops a print job.
SpiQmClose	Closes a spooler session.
SpiQmEndDoc	Ends a print job.
SpiQmOpen	Opens a spooler session.
SpiQmStartDoc	Starts a print job.
SpiQmWrite	Writes data into a print job.
SpiQueryDevice	Retrieves information about a print device.
SpiQueryJob	Retrieves information about a print job.
SpiQueryQueue	Retrieves information about a print queue.
SpiReleaseJob	Releases a print job in a print queue.
SpiReleaseQueue	Releases a print queue.

<i>Table 18-2 (Page 2 of 2). Printing and Spooler Functions</i>	
Function Name	Description
SplSetDevice	Sets information about a print device.
SplSetJob	Sets information about a print job.
SplSetQueue	Sets information about a print queue.

Table 18-3 summarizes the data structures used by the functions that print jobs, manage jobs, and manipulate the spooler configuration.

<i>Table 18-3. Printing and Spooler Structures</i>	
Structure Name	Description
DEVOPENSTRUC	Device context open data structure.
DRIVPROPS	Printer driver properties structure.
DRIVDATA	Driver data structure (for job properties).
HCINFO	Hardcopy capabilities structure.
PRDINFO3	Print device information structure (level 3).
PRDRIVINFO	Printer driver information structure.
PRINTERINFO	Print destination information structure.
PRJINFO2	Print job information structure (level 2).
PRJINFO3	Print job information structure (level 3).
PRPORTINFO	Port information structure (level 0).
PRPORTINFO1	Port information structure (level 1).
PRQINFO3	Print queue information structure (levels 3 and 4).
PRQINFO6	Print-queue information structure (level 6).
PRQPROCINFO	Queue processor information structure.
QMOPENSTRUC	Spooler open data structure.

Appendixes

Appendix A. Matrix Multiplication

To show how matrix multiplication is implemented, here are two matrixes:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

The multiplication of these two matrixes produces a value for each of the nine elements of the resulting matrix:

$$\begin{bmatrix} \text{element 1} & \text{element 2} & \text{element 3} \\ \text{element 4} & \text{element 5} & \text{element 6} \\ \text{element 7} & \text{element 8} & \text{element 9} \end{bmatrix}$$

To produce element 1 of the matrix, element a is multiplied by element j, element b is multiplied by element m, and element c is multiplied by element p. That is:

$$\text{element 1} = (a \times j) + (b \times m) + (c \times p)$$

To produce element 2 of the matrix, element a is multiplied by element k, element b is multiplied by element n, and element c is multiplied by element q. To produce element 3 of the matrix, elements a, b, and c are multiplied by their corresponding elements (l, o, and r) in the third vertical line of the second matrix. To produce element 4 of the matrix, you move down a row in the first matrix. That is, element d is multiplied by element j, element e is multiplied by element m, and element f is multiplied by element p. You continue the multiplication in this way until each of the nine elements has a value. The complete workings for this example are as follows:

$$\text{element 1} = (a \times j) + (b \times m) + (c \times p)$$

$$\text{element 2} = (a \times k) + (b \times n) + (c \times q)$$

$$\text{element 3} = (a \times l) + (b \times o) + (c \times r)$$

$$\text{element 4} = (d \times j) + (e \times m) + (f \times p)$$

$$\text{element 5} = (d \times k) + (e \times n) + (f \times q)$$

$$\text{element 6} = (d \times l) + (e \times o) + (f \times r)$$

$$\text{element 7} = (g \times j) + (h \times m) + (i \times p)$$

$$\text{element 8} = (g \times k) + (h \times n) + (i \times q)$$

$$\text{element 9} = (g \times l) + (h \times o) + (i \times r)$$

Note that if the order of the two matrixes is reversed, the results of the multiplication are different.

Here is a simple example in which an object is scaled by a factor of 3, and then translated by (5,4):

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 5 & 4 & 1 \end{bmatrix}$$

You can multiply together as many transformation matrixes as you require.

Appendix B. GPI Functions

A number of the GPI error conditions indicate that a function has been used in the wrong context. This appendix lists every function and shows, for each one, whether it can be used:

- In a micro presentation space
- While there is an open segment bracket
- While there is an open area bracket
- While there is an open element bracket
- While there is an open path bracket.

A check mark (✓) means that a function can be used; a cross (X) means that it cannot. There are some additional qualifiers in the form of superscript numbers. These generally indicate some further restriction on the context in which a function can be called, and are explained at the end of this appendix on page B-9.

Table B-1 (Page 1 of 7). Where GPI Functions Can Be Called					
GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiAnimatePalette	√ ^{8,9}	√	X	√	X
GpiAssociate	X	√	X ⁷	√ ⁶	X ⁷
GpiBeginArea	√	√	X	√	X
GpiBeginElement	X	√	√	X	√
GpiBeginPath	√	√	X	√	X
GpiBitBlt	√	√	X	√	X
GpiBox	√	√	√	√	√
GpiCallSegmentMatrix	X	√	√	√	√
GpiCharString	√	√	X	√	√
GpiCharStringAt	√	√	X	√	√
GpiCharStringPos	√	√	X	√	√
GpiCharStringPosAt	√	√	X	√	√
GpiCloseFigure	√	√	X	√	√
GpiCloseSegment	X	√	X ⁷	√	X ⁷
GpiCombineRegion	√	√	X	√	X
GpiComment	√	√	√	√	√
GpiConvert	√	√	√	√	√
GpiConvertWithMatrix	√	√	√	√	√
GpiCopyMetaFile	√	√	√	√	√
GpiCorrelateChain	X	√	X	X	X
GpiCorrelateFrom	X	√	X	X	X
GpiCorrelateSegment	X	√	X	X	X
GpiCreateBitmap	√	√	√	√	√
GpiCreateLogColorTable	√	√	X	√	X
GpiCreateLogFont	√	√	X	X	√
GpiCreatePalette	√	√	√	√	√
GpiCreatePS	—	—	—	—	—
GpiCreateRegion	√	√	X	√	X
GpiDeleteBitmap	√	√	√	√	√
GpiDeleteElement	X	√ ³	√ ³	X	√ ³
GpiDeleteElementRange	X	√ ³	√ ³	X	√ ³
GpiDeleteElementsBetweenLabels	X	√ ³	√ ³	X	√ ³
GpiDeleteMetaFile	√	√	√	√	√
GpiDeletePalette	√ ⁹	√	√	√	√
GpiDeleteSegment	X	√	√	√	√
GpiDeleteSegments	X	√	√	√	√
GpiDeleteSetId	√	√	X	√	√

Table B-1 (Page 2 of 7). Where GPI Functions Can Be Called

GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiDestroyPS	√ ¹	√	√	√	√
GpiDestroyRegion	√	√	X	√	X
GpiDrawBits	√ ¹⁰	√	√	√	√
GpiDrawChain	X	√	X	X	X
GpiDrawDynamics	X	√	X	X	X
GpiDrawFrom	X	√	X	X	X
GpiDrawSegment	X	√	X	X	X
GpiElement	X	√	√	X	√
GpiEndArea	√	√	√	√	X
GpiEndElement	X	√	√	√	√
GpiEndPath	√	√	X	√	√
GpiEqualRegion	√	√	X	√	X
GpiErase	√	√	X	√	X
GpiErrorSegmentData	X	√	√	√	√
GpiExcludeClipRectangle	√	√	X	√	X
GpiFillPath	√	√	X	√	X
GpiFullArc	√	√	√	√	√
GpiGetData	X	√ ⁵	√	√	√
GpiImage	√	√	X	√	X
GpiIntersectClipRectangle	√	√	X	√	X
GpiLabel	X	√	√	X	√
GpiLine	√	√	√	√	√
GpiLoadBitmap	√	√	√	√	√
GpiLoadFonts	√	√	√	√	√
GpiLoadMetaFile	√	√	√	√	√
GpiLoadPublicFonts	√	√	√	√	√
GpiMarker	√	√	X	√	√
GpiModifyPath	√	√	X	√	X
GpiMove	√	√	√	√	√
GpiOffsetClipRegion	√	√	X	√	X
GpiOffsetElementPointer	X	√ ³	√ ³	X	√ ³
GpiOffsetRegion	√	√	X	√	X
GpiOpenSegment	X	X	X ⁷	√ ⁶	X ⁷
GpiOutlinePath	√	√	X	√	X
GpiPaintRegion	√	√	X	√	X
GpiPartialArc	√	√	√	√	√
GpiPathToRegion	√	√	√	√	X

Table B-1 (Page 3 of 7). Where GPI Functions Can Be Called

GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiPlayMetaFile	✓	X	X	X	X
GpiPointArc	✓	✓	✓	✓	✓
GpiPolyFillet	✓	✓	✓	✓	✓
GpiPolyFilletSharp	✓	✓	✓	✓	✓
GpiPolygons	✓	✓	X	✓	X
GpiPolyLine	✓	✓	✓	✓	✓
GpiPolylineDisjoint	✓	✓	✓	✓	✓
GpiPolyMarker	✓	✓	X	✓	✓
GpiPolySpline	✓	✓	✓	✓	✓
GpiPop	X	✓	✓	✓	✓
GpiPtInRegion	✓	✓	X	✓	X
GpiPtVisible	✓	✓	X	✓	X
GpiPutData	X	✓	✓	✓	✓
GpiQueryArcParams	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryAttrMode	X	✓	✓	✓	✓
GpiQueryAttrs	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryBackColor	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryBackMix	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryBitmapBits	✓	✓ ⁴	X	✓	X
GpiQueryBitmapDimension	✓	✓	✓	✓	✓
GpiQueryBitmapHandle	✓	✓	✓	✓	✓
GpiQueryBitmapInfoHeader	✓	✓	✓	✓	✓
GpiQueryBitmapParameters	✓	✓	✓	✓	✓
GpiQueryBoundaryData	✓	✓	✓	✓	✓
GpiQueryCharAngle	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharBox	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharBreakExtra	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharDirection	✓	✓	X	✓	✓
GpiQueryCharExtra	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharMode	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharSet	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharShear	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharStringPos	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryCharStringPosAt	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryClipBox	✓	✓	✓	✓	✓
GpiQueryClipRegion	✓	✓	✓	✓	✓
GpiQueryColor	✓	✓ ²	✓ ²	✓ ²	✓ ²

Table B-1 (Page 4 of 7). Where GPI Functions Can Be Called

GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiQueryColorData	✓	✓	✓	✓	✓
GpiQueryColorIndex	✓	✓	✓	✓	✓
GpiQueryCp	✓	✓	✓	✓	✓
GpiQueryCurrentPosition	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryDefArcParams	✓	✓	✓	✓	✓
GpiQueryDefAttrs	✓	✓	✓	✓	✓
GpiQueryDefaultViewMatrix	✓	✓	✓	✓	✓
GpiQueryDefCharBox	✓	✓	✓	✓	✓
GpiQueryDefTag	✓	✓	✓	✓	✓
GpiQueryDefViewingLimits	✓	✓	✓	✓	✓
GpiQueryDevice	✓	✓	✓	✓	✓
GpiQueryDeviceBitmapFormats	✓	✓	✓	✓	✓
GpiQueryDrawControl	✓	✓	✓	✓	✓
GpiQueryDrawingMode	X	✓	✓	✓	✓
GpiQueryEditMode	X	✓	✓	✓	✓
GpiQueryElement	X	✓ ³	✓ ³	✓ ³	✓ ³
GpiQueryElementPointer	X	✓ ³	✓ ³	✓ ³	✓ ³
GpiQueryElementType	X	✓ ³	✓ ³	✓ ³	✓ ³
GpiQueryFaceString	✓	✓	✓	✓	✓
GpiQueryFontAction	✓	✓	✓	✓	✓
GpiQueryFontMetrics	✓	✓	✓	✓	✓
GpiQueryFonts	✓	✓	✓	✓	✓
GpiQueryFullFontFileDescriptions	✓	✓	✓	✓	✓
GpiQueryGraphicsField	✓	✓	✓	✓	✓
GpiQueryInitialSegmentAttrs	X	✓	✓	✓	✓
GpiQueryKerningPairs	✓	✓	✓	✓	✓
GpiQueryLineEnd	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryLineJoin	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryLineType	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryLineWidth	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryLineWidthGeom	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryLogColorTable	✓	✓	✓	✓	✓
GpiQueryLogicalFont	✓	✓	✓	✓	✓
GpiQueryMarker	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryMarkerBox	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryMarkerSet	✓	✓ ²	✓ ²	✓ ²	✓ ²
GpiQueryMetaFileBits	✓	✓	✓	✓	✓

Table B-1 (Page 5 of 7). Where GPI Functions Can Be Called

GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiQueryMetaFileLength	✓	✓	✓	✓	✓
GpiQueryMix	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryModelTransformMatrix	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryNearestColor	✓	✓	✓	✓	✓
GpiQueryNumberSetIds	✓	✓	✓	✓	✓
GpiQueryPageViewport	✓	✓	✓	✓	✓
GpiQueryPalette	✓	✓	✓	✓	✓
GpiQueryPaletteInfo	✓	✓	✓	✓	✓
GpiQueryPattern	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryPatternRefPoint	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryPatternSet	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryPel	✓	✓	✓	✓	✓
GpiQueryPickAperturePosition	✓	✓	✓	✓	✓
GpiQueryPickApertureSize	✓	✓	✓	✓	✓
GpiQueryPS	✓	✓	✓	✓	✓
GpiQueryRealColors	✓	✓	✓	✓	✓
GpiQueryRegionBox	✓	✓	✓	✓	✓
GpiQueryRegionRects	✓	✓	✓	✓	✓
GpiQueryRGBColor	✓	✓	✓	✓	✓
GpiQuerySegmentAttrs	X	✓	✓	✓	✓
GpiQuerySegmentNames	X	✓	✓	✓	✓
GpiQuerySegmentPriority	X	✓	✓	✓	✓
GpiQuerySegmentTransformMatrix	X	✓	✓	✓	✓
GpiQuerySetIds	✓	✓	✓	✓	✓
GpiQueryStopDraw	X	✓	✓	✓	✓
GpiQueryTag	X	√ ²	√ ²	√ ²	√ ²
GpiQueryTextAlignment	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryTextBox	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryViewingLimits	✓	√ ²	√ ²	√ ²	√ ²
GpiQueryViewingTransformMatrix	X	✓	✓	✓	✓
GpiQueryWidthTable	✓	✓	✓	✓	✓
GpiRectInRegion	✓	✓	X	✓	X
GpiRectVisible	✓	✓	X	✓	X
GpiRemoveDynamics	X	✓	X	X	X
GpiResetBoundaryData	✓	✓	✓	✓	✓
GpiResetPS	✓	✓	√ ⁶	√ ⁶	√ ⁶
GpiRestorePS	✓	√ ⁴	X	√ ⁴	X

Table B-1 (Page 6 of 7). Where GPI Functions Can Be Called

GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiRotate	✓	✓	✓	✓	✓
GpiSaveMetaFile	✓	✓	✓	✓	✓
GpiSavePS	✓	✓ ⁴	X	✓ ⁴	X
GpiScale	✓	✓	✓	✓	✓
GpiSelectPalette	✓	✓	X	✓	✓
GpiSetArcParams	✓	✓	✓	✓	✓
GpiSetAttrMode	X	✓	✓	✓	✓
GpiSetAttrs	✓	✓	X	✓	✓
GpiSetBackColor	✓	✓	X	✓	X
GpiSetBackMix	✓	✓	X	✓	X
GpiSetBitmap	✓	✓	X	✓	X
GpiSetBitmapBits	✓	✓	X	✓	X
GpiSetBitmapDimension	✓	✓	✓	✓	✓
GpiSetBitmapId	✓	✓	✓	✓	✓
GpiSetCharAngle	✓	✓	X	✓	✓
GpiSetCharBox	✓	✓	X	✓	✓
GpiSetCharBreakExtra	✓	✓	✓	✓	✓
GpiSetCharDirection	✓	✓	X	✓	✓
GpiSetCharExtra	✓	✓	✓	✓	✓
GpiSetCharMode	✓	✓	X	✓	✓
GpiSetCharSet	✓	✓	X	✓	✓
GpiSetCharShear	✓	✓	X	✓	✓
GpiSetClipPath	✓	✓	X	✓	X
GpiSetClipRegion	✓	✓	X	✓	X
GpiSetColor	✓	✓	X	✓	✓
GpiSetCp	✓	✓	X	✓	✓
GpiSetCurrentPosition	✓	✓	✓	✓	✓
GpiSetDefArcParams	✓	✓	✓	✓	✓
GpiSetDefAttrs	✓	✓	✓	✓	✓
GpiSetDefaultViewMatrix	✓	✓	X	✓	X
GpiSetDefTag	✓	✓	✓	✓	✓
GpiSetDefViewingLimits	✓	✓	✓	✓	✓
GpiSetDrawControl	✓	X	X	X	X
GpiSetDrawingMode	X	X	X	X	X
GpiSetEditMode	X	✓	✓	X	✓
GpiSetElementPointer	X	✓ ²	✓ ²	X	✓ ²
GpiSetElementPointerAtLabel	X	✓ ²	✓ ²	X	✓ ²

<i>Table B-1 (Page 7 of 7). Where GPI Functions Can Be Called</i>					
GPI Function	Micro PS	Segment Bracket	Area Bracket	Element Bracket	Path Bracket
GpiSetGraphicsField	✓	✓	X	✓	X
GpiSetInitialSegmentAttrs	X	✓	✓	✓	✓
GpiSetLineEnd	✓	✓	X	✓	✓
GpiSetLineJoin	✓	✓	X	✓	✓
GpiSetLineType	✓	✓	X	✓	✓
GpiSetLineWidth	✓	✓	X	✓	✓
GpiSetLineWidthGeom	✓	✓	X	✓	X
GpiSetMarker	✓	✓	X	✓	✓
GpiSetMarkerBox	✓	✓	X	✓	✓
GpiSetMarkerSet	✓	✓	X	✓	✓
GpiSetMetaFileBits	✓	✓	✓	✓	✓
GpiSetMix	✓	✓	X	✓	✓
GpiSetModelTransformMatrix	✓	✓	✓	✓	✓
GpiSetPageViewport	✓	✓	X	✓	X
GpiSetPaletteEntries	✓ ^B	✓	X	✓	✓
GpiSetPattern	✓	✓	X	✓	X
GpiSetPatternRefPoint	✓	✓	X	✓	X
GpiSetPatternSet	✓	✓	X	✓	X
GpiSetPel	✓	✓	X	✓	X
GpiSetPickAperturePosition	✓	✓	X	✓	X
GpiSetPickApertureSize	✓	✓	X	✓	X
GpiSetPS	✓	✓	✓	✓	✓
GpiSetRegion	✓	✓	X	✓	X
GpiSetSegmentAttrs	X	✓	✓	✓	✓
GpiSetSegmentPriority	X	X ⁵	✓	X	✓
GpiSetSegmentTransformMatrix	X	✓	✓	✓	✓
GpiSetStopDraw	X	✓	✓	✓	✓
GpiSetTag	X	✓	X	✓	✓
GpiSetTextAlignment	✓	✓	✓	✓	✓
GpiSetViewingLimits	✓	✓	X	✓	X
GpiSetViewingTransformMatrix	X	X	X	✓	X
GpiStrokePath	✓	✓	X	✓	X
GpiTranslate	✓	✓	✓	✓	✓
GpiUnloadFonts	✓	✓	✓	✓	✓
GpiUnloadPublicFonts	✓	✓	✓	✓	✓
GpiUnrealizeColorTable	✓	✓	X	✓	X
GpiWCBitBit	✓	✓	✓	✓	✓

Notes:

1. Not valid to a cached micro presentation space.
2. Valid only when the actual drawing mode is draw or draw-and-retain. The actual drawing mode is determined as shown in the following table:

<i>Table B-2. The Current Drawing Mode</i>			
GpiSetDrawingMode parameter	Context		
	Chained Segment	Unchained Segment	Outside Segment
DM_DRAWANDRETAIN	draw-and-retain	retain	draw
DM_RETAIN	retain	retain	draw
DM_DRAW	draw	retain	draw

For example, if the current drawing mode parameter is DM_RETAIN, and primitives are being drawn outside a segment, then the actual drawing mode is draw.

3. Valid only when the actual drawing mode (see note 2) is retain.
4. Valid only when the actual drawing mode (see note 2) is draw.
5. Not valid if the specified segment is the current open segment.
6. Bracket (path, element, or area) is ended without error.
7. Severity is Warning.
8. If associated with a metafile, only the final values are recorded in the metafile.
9. Palette must not be current.
10. Device context must be able to support bit map operations.

Appendix C. Graphics Attributes

Every graphics presentation space has a set of *graphics attributes*. A normal presentation space has a larger set of graphics attributes than a micro or cached micro presentation space. (Segment-related attributes, for example, do not apply in micro presentation spaces.) These attributes all have default values, which means that they always have an effect on graphics, even if you have not explicitly specified their values. The attributes can be broken into two general groups. The first group comprises those attributes that form a part of the picture and that can vary as the picture is drawn. These are:

- All primitive attributes
- The segment attributes
- The primitive tag
- The current position
- The viewing window
- The clipping path
- The model and segment transformations
- The viewing transformation.

With the exception of the segment attributes and the viewing transformation, these attributes are reset to their default values at the start of a root segment, unless the fast-chaining attribute is set. If the fast-chaining attribute is set, their values cannot be guaranteed. You should, therefore, explicitly set any attribute values required in the segment. Similarly, the values of these attributes cannot be guaranteed following a call to:

Any GpiDraw function
Any GpiCorrelate function
GpiCallSegmentMatrix
GpiCloseSegment.

Therefore, if any of these functions is followed by primitives outside a segment, you should explicitly set required attribute values. When GpiCloseSegment is followed by GpiOpenSegment (and also between any two segments in the chain), the fast-chaining attribute determines what happens to the current values of these attributes. The viewing transformation and the segment attributes are unaffected by fast-chaining.

These attributes take effect when graphics are sent to an output device, *not* when graphics are defined. For this reason, the calls to GpiQuery that retrieve the current values of these attributes are invalid in retain mode. The majority of the group-one attribute-setting functions cause graphics orders to be added to the current segment.

The second group of attributes defines the environment in which the picture is drawn. These attributes do not normally vary as the picture is drawn, but have an overall effect on the result of any drawing or correlation operation. This group includes:

- The default viewing transformation
- The page viewport
- The graphics field
- The clipping region
- The pick-aperture size
- The drawing controls.

The functions to GpiQuery that retrieve the current values of these attribute are valid in all drawing modes. None of the group-two attribute-setting functions causes graphics orders to be added to the current segment.

The following table lists the graphics attributes, identifies the GPI function or functions that you use to change current settings, and lists the default value of each one. Note that these default values are the initial settings. The table also indicates whether the attribute is a group-one or a group-two attribute.

Graphics Attribute	GPI Function	Default Value	Group One	Group Two
Arc parameters	GpiSetArcParams	(1,1,0,0)	✓	
Foreground color	GpiSetColor ¹	CLR_DEFAULT (device-dependent)	✓	
Foreground mix	GpiSetMix ¹	FM_DEFAULT (overpaint)	✓	
Background color	GpiSetBackColor ¹	CLR_DEFAULT (device-dependent)	✓	
Background mix	GpiSetBackMix ¹	BM_DEFAULT (leave-alone)	✓	
Character angle	GpiSetCharAngle ¹	(1,0)	✓	
Character box	GpiSetCharBox ¹	Device-dependent	✓	
Character Direction	GpiSetCharDirection ¹	CHDIRN_DEFAULT (left to right)	✓	
Character mode	GpiSetCharMode ¹	CM_DEFAULT (mode 1)	✓	
Character set	GpiSetCharSet ¹	LCID_DEFAULT (system font)	✓	
Character shear	GpiSetCharShear ¹	(0,1)	✓	
Line end	GpiSetLineEnd ¹	LINEEND_DEFAULT (flat)	✓	
Line join	GpiSetLineJoin ¹	LINEJOIN_DEFAULT (beveled)	✓	
Line type	GpiSetLineType ¹	LINETYPE_DEFAULT (solid)	✓	
Line width	GpiSetLineWidth ¹	LINEWIDTH_DEFAULT (1.0)	✓	
Geometric line width	GpiSetLineWidthGeom ¹	1	✓	
Marker	GpiSetMarker ¹	MARKSYM_DEFAULT (cross)	✓	
Marker box	GpiSetMarkerBox ¹	Device-dependent	✓	
Marker set	GpiSetMarkerSet ¹	LCID_DEFAULT (system marker set)	✓	
Pattern	GpiSetPattern ¹	PATSYM_DEFAULT (solid)	✓	
Pattern reference point	GpiSetPatternRefPoint ¹	(0,0)	✓	
Pattern set	GpiSetPatternSet ¹	LCID_DEFAULT (system pattern set)	✓	
Model transformation	GpiSetModelTransformMatrix	Identity	✓	
Instance transformation	GpiCallSegmentMatrix	Identity	✓	
Segment transformation	GpiSetSegmentTransformMatrix	Identity	✓	

Graphics Attribute	GPI Function	Default Value	Group One	Group Two
Viewing transformation	GpiSetViewingTransformMatrix	Identity	✓	
Default viewing transformation	GpiSetDefaultViewMatrix	Identity		✓
Page viewport	GpiSetPageViewport	Device-dependent		✓
Clipping path	GpiSetClipPath	No clipping	✓	
Viewing window	GpiSetViewingLimits	No clipping	✓	
Graphics field	GpiSetGraphicsField	No clipping		✓
Clipping region	GpiSetClipRegion	No clipping		✓
Tag	GpiSetTag	0	✓	
Segment attributes	GpiSetInitialSegmentAttrs GpiSetSegmentAttrs	ATTR_DETECTABLE - OFF ATTR_VISIBLE - ON ATTR_CHAINED - ON ATTR_DYNAMIC - OFF ATTR_FASTCHAIN - ON ATTR_PROP_DETECTABLE - ON ATTR_PROP_VISIBLE - ON	✓	
Pick-aperture size	GpiSetPickApertureSize	Device-dependent		✓
Current position	GpiSetCurrentPosition GpiMove ²	(0,0)	✓	
Drawing controls	GpiSetDrawControl	DCTL_ERASE - OFF DCTL_DISPLAY - ON DCTL_BOUNDARY - OFF DCTL_DYNAMIC - OFF DCTL_CORRELATE - OFF		✓
Notes: ¹ Can also be set using GpiSetAttrs. ² Is also updated indirectly by most primitive-drawing functions.				

Each of the following functions causes all group-one and group-two graphics attributes to be set to their default values:

- GpiCreatePS
- GpiSetPS
- GpiResetPS, if GRES_ALL or GRES_SEGMENTS is specified
- WinGetPS
- WinGetScreenPS
- WinBeginPaint, with a NULL presentation space handle.

In specific circumstances, some of the GPI functions modify the group-one attributes and thus make their values unpredictable. Therefore, when you call any of these functions, you should respecify attribute values that have a particular importance to your application. For example, if the current foreground color is CLR_RED before you call GpiDrawChain, you cannot always rely on the current color remaining CLR_RED when GpiDrawChain completes. If you want to continue working in red, respecify the color when GpiDrawChain completes. In general, the functions that affect group-one-attribute values are those related to the drawing and correlation of retained graphics, and to the creation, closing, and deletion of graphics segments.

With the exception of the viewing transformation and the segment attributes, the default values of these attributes apply to all primitive-drawing that occurs outside a segment bracket.

- The following function sets the group-one attributes to their default values, and in addition sets the current clipping region and page viewport to their default values.
 - GpiAssociate.
- Each of the following functions makes the group-one-attribute values unpredictable:
 - GpiCloseSegment
 - GpiCorrelateChain
 - GpiCorrelateFrom
 - GpiCorrelateSegment
 - GpiErase
 - GpiCallSegmentMatrix
 - GpiDrawDynamics
 - GpiRemoveDynamics.¹
- Each of the following functions makes group-one-attribute values unpredictable if a segment to be deleted is open when the function is called:
 - GpiDeleteSegment
 - GpiDeleteSegments.
- Each of the following functions makes group-one-attribute values unpredictable if there is no open segment when the function is called:
 - GpiDrawChain
 - GpiDrawFrom
 - GpiDrawSegment.

If there is an open segment when any of these functions is called, and that segment was the last one drawn, then group-one-attribute values are unaffected. If, however, dynamic segments were caused to be redrawn by the same function, group-one-attribute values are made unpredictable. This occurs because dynamic segments are always drawn after nondynamic segments.

¹ In the last two functions here, the foreground-mix value is always set to FM_XOR, and the background-mix value is always set to BM_LEAVEALONE.

Appendix D. Graphics Orders

In the retain and draw-and-retain drawing modes, specific GPI functions (primitive-drawing and attribute-setting functions, plus some others) cause graphics orders to be stored in the current segment. A graphics order is a sequence of one or more bytes of data that describe a graphics function. There is typically a one-to-one correspondence between a GPI function and a graphics order. You do not need to understand the various formats and contents of the graphics orders, unless:

- You are using GpiGetData or GpiPutData for bulk transfer of data that you want to edit.
- You are simply copying data from one segment to another.
- You are using GpiElement to add data to a segment, or GpiQueryElement to retrieve data from a segment.
- You are examining the contents of a metafile.

Both the graphics orders and the metafile structure are described in the *Presentation Manager Programming Reference*. This appendix describes the header file PMORD.H, which has been provided to allow you to manipulate the graphics orders more easily.

The Graphics-Orders Header File (PMORD.H)

A set of helper constants, macros, and structures has been provided to help you decode and encode graphics orders. These items are defined in the header file PMORD.H.

There are four types of graphics orders. The first byte of each order, regardless of the graphics-order type, is the order code itself, which either partially or completely describes what follows. Depending on the order type, the graphics order can contain further information.

The four types of graphics order are:

1-Byte Order

The 1-byte order comprises a single byte:

BYTE 1 : order code.

2-Byte Order

The 2-byte order consists of two bytes:

BYTE 1 : order code

BYTE 2 : associated value.

Long Order

The long order can comprise up to 257 bytes of information:

BYTE 1 : order code

BYTE 2 : length of order (0 to 255)

BYTE 3-257 : associated value bytes
depending on the order code.

There is a special long order (Escape) where:

BYTE 3 : escape type

BYTE 4 : escape identifier

BYTE 5-257 : associated value bytes
depending on the escape
identifier.

Very Long Order

The very long order can comprise up to 65537 bytes of information:

BYTE 1 : order code

BYTE 2 : order qualifier

BYTE 3 : length of order
(most significant byte)

BYTE 4 : length of order
(least significant byte -
length of order is 0 to 65535)

BYTE 5-65537 : associated value bytes
depending on the order
qualifier.

There is a special very long order (Escape) where :

BYTE 5 : escape type

BYTE 6 : escape identifier

BYTE 7-65537 : associated value bytes
depending on the escape
identifier.

Decoding Graphics Orders

The recommended way of decoding a buffer of graphics orders (in C language) is to use a pointer to address the first byte of the buffer, and then retrieve the graphics order it contains. To discover which of the four types of order you have, use the following macros:

- `BYTE_ORDER` (1-byte order)
- `SHORT_ORDER` (2-byte order)
- `LONG_ORDER` (long order)
- `VLONG_ORDER` (very long order).

These macros are defined in the header file `PMORD.H`. Each macro processes a single byte of data and returns a Boolean value (zero or nonzero). A zero value means that the order is not of that type. When you know the graphics-order type, you can establish the length of the order, and add the length to the pointer. You can then retrieve the next order in the buffer, and repeat the process until all data has been retrieved.

You can decode the graphics-order data itself by providing a routine for each of the order types, or a routine for each individual order:

- For a 1-byte graphic order, the decoding routine should simply return a length of 1.
- For a 2-byte graphic order, the decoding routine can use the overlay structure `ORDER` to decode the data. The routine should return a length of 2.
- For a long order, the decoding routine can use the overlay structure `LORDER` to decode the data. The length of the data is a variable value.
- For a very long order, the decoding routine can use the overlay structure `VORDER` to decode the data. The length of the data is a variable value.

The overlay structures `ORDER`, `LORDER`, and `VORDER` are defined in the header file `PMORD.H`.

You can build graphics orders using the same structures and order types that are used for decoding graphics orders.

Naming Conventions

The names of the graphics-order codes are in the form `OCODE_Gxxx`. The `Gxxx` abbreviation is the name of the individual order, and can be used for types, structures, and constants directly related to that order. In the header file, there is a comment on the same line as each of the orders that describes the order. For example, the Begin Area order (GBAR) is described in the header file as follows:

```
#define OCODE_GBAR    0x68 /* Begin area    */
#define GBAR_BOUNDARY 0xC0
```

Notes:

1. In some structures, an S or an L is added to the name to differentiate between the short-coordinate form (16-bit) and the long-coordinate form (32-bit). For example, the Set Arc Parameters order (GSAP) is as follows:

```
#define OCODE_GSAP    0x22
#define OCODE_GPSAP   0x62

typedef struct _ORDERS_GSAP {
    SHORT  p;
    SHORT  q;
    SHORT  r;
    SHORT  s;
} ORDERS_GSAP;

typedef struct _ORDERL_GSAP {
    LONG   p;
    LONG   q;
    LONG   r;
    LONG   s;
} ORDERL_GSAP;
```

In this example, the structures `ORDERS_GSAP` and `ORDERL_GSAP` are shared by GSAP (set arc parameters) and GPSAP (push and set arc parameters). As a rule, there is structure sharing between the set and push-and-set forms of graphics orders.

2. There is structure-sharing between the current-position and the given-position forms of some orders. For example, the orders GCARC (arc at current position) and GARC (arc at given position) share a structure.

Index

A

- a-space of a character 9-10, 9-13
- abGeneralData field 18-14
- about coordinate spaces 17-1
- about line and arc primitives 3-1
- about paths 10-1
- about print subsystem components 18-1
- about regions 11-1
- about transformation functions 17-22
- about transformation operations 17-14
- about transformations 17-4
- accumulating transformations 17-22
- actual drawing mode 12-6, 13-2
- addressing specific graphics element 13-5
- Adobe fonts 9-4
- aligning text 6-13, 6-14, 6-20
- allocating storage 18-13
- alternate and winding fill modes, example 10-11
- alternate mode, area construction 5-10
- alternate mode, description 5-10
- alternate pels on 3-4
- AM_NOPRESERVE 2-5, 12-9
- AM_PRESERVE 2-5, 3-2, 12-9, 14-4
- application design considerations for printing 18-22
- application interface and data flow, printing 18-7
- application printer setup dialog 18-13
- applying transformation functions 17-27
- applying transformation operations directly to the transformation matrix 17-27
- applying transformations other than the identity transformation 17-7
- applying transformations to root segments 17-24
- arc primitives
 - about 3-1
 - attributes 1-3, 3-10
 - Bezier splines 3-19
 - circles 3-10
 - color and mix attributes 3-5
 - current arc 3-10
 - defining a circle 3-14
 - defining an ellipse 3-12
 - description 1-3, 3-1
 - drawing direction 3-10
 - ellipses 3-10
 - fillets 3-10
 - full arcs 3-12
 - functions that draw simple arcs 3-12
 - long values 3-14
 - multiple arcs 3-10
 - multiple-arc primitive family 3-17
 - orientation 3-10
 - partial arcs 3-12, 3-14
 - shape 3-10
- arc primitives (*continued*)
 - sharpness of a fillet 3-18
 - simple-arc primitive family 3-12
 - size 3-10
 - splines 3-10
 - tilted ellipse 3-14
 - transformation matrix 3-11
 - unit circle 3-10
 - used for outline fonts 6-2
 - using 3-20
 - 3-point arcs 3-12
- ARCPARAMS 3-10, 3-27
- ARE parameter 18-27
- area attributes for paths 10-6
- area background color 11-3
- area boundaries 5-9
- area boundary 5-9
- area bracket attributes 5-9
- area brackets 5-7
- area color and mix attributes 5-5
- area constructed in different directions, winding mode 5-11
- area construction 5-9, 5-10
- area foreground color 11-3
- area foreground mix 11-3
- area primitive
 - functions valid in B-1
- area primitives
 - about 5-1
 - area brackets 5-7
 - area constructed in different directions 5-11
 - attribute default values 5-2
 - attributes 1-3, 5-1
 - boundaries 5-9
 - color and mix attributes 5-5
 - colors 5-5
 - construction 5-10
 - customizing pattern sets 5-4
 - default pattern set 5-4
 - description 1-3
 - example of an area 5-1
 - pattern symbol attribute 5-2
 - summary of functions 5-20
 - using 5-15
 - winding mode, construction 5-11
- AREABUNDLE 1-3, 3-10, 5-6, 5-9, 5-13, 5-20, 7-25, 8-20, 10-2, 11-3
- area, description 5-1
- aspect ratios 8-11
- associating
 - correlation tags with primitives 14-4
 - presentation space with device context 18-18, 18-21
 - presentation spaces with device contexts 1-2, 2-1, 2-10

association, between presentation spaces and device contexts 2-10

asynchronous drawing thread restrictions 15-6

attribute currentness 5-9

attribute default values, area primitives 5-2

attributes

arc primitive 1-3, 3-10

area background color 11-3

area for paths 10-6

area foreground color 11-3

area foreground mix 11-3

area primitive 1-3

background mix attribute 12-8

boundary 5-9

box 4-3

bracket 5-9

bundles 2-5

changing defaults 12-4

character string primitive 1-4

character string primitives 6-2

color 3-2

color and mix 1-4, 5-5

color, default 3-2

construction 5-9

cosmetic line for paths 10-6

currentness 2-5, 5-9, 5-12, 12-2

default mix 3-2

default pattern set 5-4

description 5-1

differences between primitive and segment

attributes 12-4

end 3-2

for areas

for character strings

for lines and arcs

for markers

for paths

foreground mix attribute 12-8

geometric lines 3-3

geometric width 3-2

geometric width, default 3-2

initial segment 12-12, 12-13

join 3-2

line color and mix 3-5

line end 10-4

line ends, standard 10-4

line join 10-5

line primitive 1-3

LINETYPE_DOT 1-3

LINEWIDTH_DEFAULT 3-3

LINEWIDTH_NORMAL 3-3

LINEWIDTH_THICK 3-3

marker color and mix 4-4

marker primitive 1-4

mix 3-2, 7-15

modes 12-9

AM_PRESERVE 3-2

of fonts 9-4

attributes (*continued*)

path 10-2

path color and mix 10-6

path default values 10-2

pattern reference point 5-3, 11-3

pattern set 5-4, 11-3

pattern symbol 11-3

preventing the resetting of attributes 12-4

primitive 1-3

primitive, description 2-5

push-and-set mode 12-9

region 11-2

set 4-4

set mode 12-9

symbols 4-2

type 3-2

type, default 3-2

unreliable after segment closing 12-4

values 2-5

width 3-2

width, default 3-2

with ON and OFF settings 12-4

ATTR_CHAIN 12-5

ATTR_CHAINED 12-4, 12-5

ATTR_DETECTABLE 14-3

ATTR_DYNAMIC 14-3

ATTR_PROP_DETECTABLE 14-3

average character width 9-13

B

b-space of a character 9-10, 9-13

background color 3-5, 5-1, 6-16, 7-9, 13-14

background colors 4-4, 5-5

background mix 4-4, 5-5, 7-13

background mix attribute 5-1, 6-16, 12-8

background mix, area 5-6

background mix, marker 4-4

bar graph drawn with line and arc primitives 3-1

base pattern set, table 5-2

base printing, description 18-7

baselines 6-8, 6-9, 6-12

BA_ALTERNATE construction mode 5-10

BA_BOUNDARY value 5-9

BA_NOBOUNDARY value 5-9

BA_WINDING construction mode 5-10

BBO_AND 8-12

BBO_IGNORE 8-12

BBO_OR 8-12

Bezier splines 3-19

bit counts 8-7

bit map handle 8-4

bit map planes 8-7

bit maps 8-1

amount of memory occupied 8-2

as area-fill patterns 9-25

aspect ratios 8-11

bit counts 8-7

- bit maps (*continued*)
 - bit map plane 8-7
 - changing the size of a bit map 8-12
 - color planes 8-7
 - compression options 8-12
 - conversion between color and monochrome 8-15
 - converting color bit maps to monochrome 8-15
 - copying an image from a video display 8-17
 - copying images from a display 8-15
 - create and load custom bit maps 8-6
 - creating
 - creating a custom fill pattern 8-20
 - creating custom fill pattern 5-16
 - creating with the Icon Editor
 - custom fill patterns 5-4
 - deleting 8-17
 - description 1-4
 - device dependency 8-2
 - drawing 8-20
 - handle 8-4
 - image 8-1
 - image primitive 8-9
 - information header 8-4
 - information table 8-4, 8-16
 - inverted 8-9
 - loading 8-5
 - loading from a file 8-21
 - making available to other processes 8-17
 - raster-operation (ROP) value 8-13
 - saving 8-16
 - scaling 8-20
 - single pel manipulation 8-15
 - specifying mix values 8-13
 - static appearance 8-5
 - storing color information 8-6
 - storing in a metafile 8-22
 - storing in metafiles 8-16
 - used for image fonts 6-2
 - using 8-17
 - when to use 8-2
- bit map, description 8-1
- bit-map data, transformation of 17-37
- bit-map data, transforming 17-37
- BITMAPHEADER2 8-2
- BITMAPINFOHEADER2 8-4, 8-6, 8-24
- BITMAPINFO2 8-2, 8-6, 8-24
- blueprint created with line and arc primitives 3-1
- BM_DEFAULT 7-13, 15-5
- BM_LEAVEALONE 4-5, 5-6, 7-13, 7-19, 15-5
- BM_OR 7-13
- BM_OVERPAINT 5-6, 7-13, 15-5
- BM_XOR 7-13
- boldface 6-1
- boundary data flag 16-10
- boundary determination
 - about 16-9
 - boundary data flag 16-10
 - example of a bounding rectangle 16-10

- boundary determination (*continued*)
 - in nonretained graphics 16-10
 - in retained graphics 16-10
 - using 16-11
- boundary determination, clipping 1-6
- boundary determination, definition 16-9
- boundary determination, description 1-6, 16-1
- boundary drawing control flag 13-15
- break characters 6-15
- BUNDLE data structure 7-1
- bundles 2-5
- bypassing GPI presentation layer 18-30

C

- c-space of a character 9-10, 9-13
- Cached device contexts, description 2-7
- cached micro presentation space
 - description 2-2
 - WinBeginPaint 2-3
 - WinGetPS 2-3
 - WinGetScreenPS 2-3
- CAD applications 3-1
- calculating filled paths constructed in alternate mode, example 10-11
- called segments 12-5, 12-8, 12-9, 12-13, 13-8, 14-8, 17-24, 17-29
- calling segments from other segments 17-29
- CAPS_ADDITIONAL_GRAPHICS 7-17
- CAPS_RASTER_CAPS 8-15
- CBM_INIT 8-4, 8-6
- chained segments 12-8, 12-12, 13-7, 14-4
- changing
 - attributes of segments 13-8
 - changing default attributes 12-4
 - coordinates 13-7
 - dynamic segments 13-14
 - job properties 18-14
 - position of pick apertures 14-2
 - positions of root segments 12-7
 - region definition 11-5
 - segment priority 12-7
 - size of pick apertures 14-2
- changing color tables 7-5
- character alignment
 - See text alignment
- character angle 6-8
- character cell 6-4, 6-16
- character cell default 6-7
- character clipping results for device and system fonts 18-17
- character direction 6-11, 6-12
- character fonts
 - current 9-17
 - device 9-21
 - locally defined 9-25
 - logical 9-17
 - metrics 9-18

character fonts (*continued*)

- physical 9-17
- resolution 9-22
- selecting 9-17

character mode 6-3, 9-13

See also character precision

character reference point 6-9, 9-10

character set 6-3

character shear 6-10

character string primitive 9-14

character string primitives

See also fonts

- about 6-1
- aligning text 6-20
- attributes 1-4
- background color 6-17
- background mix 6-17
- break character spacing 6-15
- changing character cells 6-7
- character angle 6-8
- character cell 6-4, 6-9, 6-16, 14-11
- character direction 6-11, 6-12
- character mode 6-3
- character reference point 6-9
- character set 6-3
- character shear 6-10
- character text alignment 6-13
- color 6-17
- color and mix 6-16
- default attributes 6-2
- default character cells 6-7
- description 1-4, 6-1
- drawing text 6-18
- extra spacing 6-15
- formatting text on a page 6-19
- horizontal alignment 6-13
- image font 6-1
- mix 6-17
- outline font 6-1
- summary of functions 6-22
- used with fonts 9-1
- using 6-18
- vertical alignment 6-14
- width vectors 6-15

CHARBUNDLE 1-4, 6-1, 6-23, 7-25

CHDIRN_BOTTOM 6-11, 6-12

CHDIRN_DEFAULT 6-12

CHDIRN_LEFTRIGHT 6-12

CHDIRN_RIGHTLEFT 6-12

CHDIRN_TOPBOTTOM 6-11, 6-12

choosing a printer 18-15

choosing fonts 18-17

circles 3-10

circle, drawing 3-22

cleaning up after a print job 18-23

clear screen drawing control flag 13-15

clearing the screen before drawing 13-14

clip path 17-3

clip path, creating 16-11

clip path, definition 16-3

clip region, adding a rectangular area 16-13

clip region, creating 16-12

clip region, description 11-1

clip region, excluding rectangular area 16-12

clip region, setting to a region intersection 16-14

clipboard 8-17

clipboard, sharing metafiles through 15-17

clipping 16-1

- about 16-1
 - adding a rectangular area to a clip region 16-13
 - area 16-1
 - boundary 11-8, 16-1
 - boundary determination 1-6
 - clip path 16-3
 - clip path, creating 16-3
 - conceptualizing the process 16-2
 - creating a clip path 16-11
 - creating a clip region 16-12
 - current clipping region 16-6
 - defining the viewing window 17-31
 - description 1-6, 10-15, 11-7
 - determining the size of a clipping area 16-14
 - device and system fonts 18-17
 - example of a clipping path 16-4
 - example of a triangular clip path 16-1
 - example of a viewing window 16-5
 - example of adding a rectangular area to a clip region 16-13
 - example of creating a clip path 16-11
 - example of creating a clip region 16-12
 - example of determining the size of a clipping area 16-14
 - example of excluding a rectangular area from a clip region 16-12
 - example of repairing picture using a clipping region 16-8
 - example of screen repairing 16-7
 - example of setting the clip region to a region intersection 16-14
 - example of the graphics field 16-5
 - excluding rectangular area from clip region 16-12
 - graphics field 16-5, 17-32
 - implementation 16-8
 - inclusive/exclusive 16-2
 - inclusive/inclusive 16-2
 - mirror functions 16-7
 - redrawing nondynamic graphics 16-8
 - setting a clip region to a region intersection 16-14
 - types of clipping areas 16-2
 - using 16-11
 - viewing pipeline 16-2
 - viewing window 16-5

clipping area 16-1

clipping areas, types 16-2

- clipping area, determining the size 16-14
- clipping boundary 16-1
- clipping process, conceptualizing 16-2
- clipping region, definition 16-6
- clipping, description 16-1
- close-segment processing 12-4
- closed figure bounded by chord and arc 3-24
- closing unattached geometric lines, example 10-5
- CLR_BACKGROUND 4-5, 5-5, 7-4, 7-5, 7-18
- CLR_BLACK 7-4
- CLR_BLUE 7-4
- CLR_BROWN 7-4
- CLR_CYAN 7-4
- CLR_DARKBLUE 7-4
- CLR_DARKCYAN 7-4
- CLR_DARKGRAY 7-4
- CLR_DARKGREEN 7-4
- CLR_DARKPINK 7-4
- CLR_DARKRED 7-4
- CLR_DEFAULT 7-4, 7-5, 7-18
- CLR_FALSE 7-4
- CLR_GREEN 7-4
- CLR_NEUTRAL 7-4, 7-5, 7-18
- CLR_PALEGRAY 7-4
- CLR_PINK 7-4
- CLR_RED 7-4
- CLR_TRUE 7-4
- CLR_WHITE 7-4, 7-18
- CLR_YELLOW 7-4
- CM_MODE1 6-4, 6-6, 6-9, 6-10
- CM_MODE2 6-4, 6-6, 6-9, 6-11
- CM_MODE3 6-4
- code page
 - selection for graphics characters 9-8
- code points 9-8
- coding the device transformation 17-36
- COL parameter 18-27
- color attributes
 - background 5-1, 6-16, 13-14
 - background of character strings 6-17
 - character string primitives 6-16
 - description 1-4
 - drawing surface 6-16
 - foreground 5-1, 6-16
 - of character strings 6-17
- color planes 8-7
- color table, default 13-14
- colors
 - about 7-1
 - available colors 7-7
 - background 4-4, 5-5, 7-10
 - background color 7-9
 - background mix 7-11, 7-13
 - changing color tables 7-5
 - color attribute 7-9
 - color on advanced display devices 7-17
 - color output and mix attributes 7-11
 - color table index mode 7-6

- colors (*continued*)
 - color tables 7-3
 - color tables in index mode 7-6
 - color tables in RGB mode 7-6
 - considerations when using monochrome displays 7-17
 - contrast color 7-18
 - creating a logical color table 7-20
 - creating a palette 7-23
 - current color 4-4, 5-5
 - default logical color table 7-4
 - defining color tables 7-5
 - determining the color-table format and index values 7-21
 - determining the index value of an RGB value 7-22
 - device independent color indexing 7-5
 - dithering 7-17
 - drawing color area fill patterns on monochrome devices 7-18
 - drawing color graphics on monochrome devices 7-18
 - drawing-surface 4-4, 5-5, 10-7
 - example of a logical color table 7-3
 - example of creating a logical color table 7-20
 - example of creating a palette 7-23
 - example of determining color-table format and index values 7-21
 - example of determining the index value of an RGB value 7-22
 - example of setting foreground color attributes 7-23
 - example of the background of a primitive 7-10
 - example of the foreground of a primitive 7-9
 - for area primitives 5-5
 - for marker primitives 4-4
 - foreground 4-4, 5-5
 - foreground color 7-9
 - foreground mix 7-11
 - GpiErase 7-18
 - GpiSetBackColor 7-10
 - implementation 7-1
 - intensity 7-2
 - investigating 7-7
 - leave-alone mix 7-16
 - loaded color tables 7-5
 - logical color table 7-2, 7-3
 - OR mix 7-14
 - output on monochrome devices 7-18
 - overpaint mix 7-14
 - palette manager 7-8
 - pel 7-1
 - phosphor 7-1
 - physical color table 7-3, 7-7
 - realizing a color palette 7-8
 - reset color 7-18
 - RGB color encoding 7-2
 - setting the primitive color attributes 7-22
 - specifying for primitives 7-10
 - the background of a primitive 7-9

- colors (*continued*)
 - the foreground of a primitive 7-9
 - using color attributes 7-20
 - XOR mix 7-15
- combining overlapping regions, example 11-5
- combining regions 11-3
- combining transformations between a coordinate space pair 17-10
- common form sizes 18-11
- comparison of paths and areas, table 10-1
- components
 - print subsystem 18-1
- concatenating transformations 17-24
- constants
 - BM_LEAVEALONE 4-5, 5-6
 - BM_OVERPAINT 5-6
 - CLR_BACKGROUND 4-5, 5-5
 - CREA_DEFAULT 15-15
 - CREA_NOREALIZE 15-15
 - CTAB_DEFAULT 15-15
 - CTAB_NOMODIFY 15-15
 - CTAB_REPLACEPALETTE 15-15
 - DDEF_DEFAULTS 15-16
 - DDEF_IGNORE 15-16
 - DDEF_LOADDISC 15-16
 - DEFAULTS 15-12
 - FM_OVERPAINT 3-5, 4-5, 5-5
 - LC_DEFAULTS 15-15
 - LC_LOADDISC 15-15
 - LC_NOLOAD 15-15
 - LINEBUNDLE 3-5
 - LT_DEFAULT 15-14
 - LT_NOMODIFY 15-14
 - LT_ORIGINALVIEW 15-14
 - PMF_COLORREALIZABLE 15-11
 - PMF_COLORTABLES 15-11, 15-12
 - PMF_DEFAULTS 15-11
 - PMF_LCID 15-12
 - PMF_LCIDS 15-11
 - PMF_LOADTYPE 15-11, 15-12
 - PMF_RESET 15-11
 - PMF_RESOLVE 15-11
 - PMF_SEGBASE 15-11
 - PMF_SUPPRESS 15-11, 15-12, 15-13
 - RES_DEFAULT 15-12
 - RES_NORESET 15-12
 - RES_RESET 15-12
 - SUP_NOSUPPRESS 15-13
 - SUP_SUPPRESS 15-13
 - TRANSFORM_ADD 17-24
- content restrictions using PM_Q_STD 18-2
- controlling region output 11-3
- controls
- conversion to clip path 10-15
- convert path to region 10-9
- convert to clip path 10-9
- converting GPI commands to PCL commands 18-4
- converting PM_Q_STD to PM_Q_RAW 18-2
- coordinate spaces
 - about 17-1
 - and transformations 17-1
 - default page 1-6
 - device 1-6, 17-4
 - media 17-4
 - model 1-6, 17-3
 - page 1-6, 17-3
 - types 1-6
 - world 17-2
- coordinate spaces and transformations 17-1
- coordinate spaces and transformations, using 17-39
- coordinate spaces, moving through 17-6
- coordinate space, world 17-1
- COP parameter 18-27
- copying regions 11-3
- correlation
 - about 14-1
 - attribute type 13-7
 - description 1-6
 - detectability attribute 13-8
 - detectable segments 14-3
 - drawing control flag 13-15, 14-2
 - during drawing 14-2
 - for called segments 14-8
 - for chained segments 14-4
 - functional sequence for retained graphics 14-5
 - hits
 - accounting for all 14-7
 - detecting 14-2
 - for called segments 14-8
 - for chained segments 14-4
 - limiting number of 14-9
 - maximum depth of, for called segments 14-8
 - maximum number 14-5
 - order of returned 14-8
 - identifiers for nonretained graphics 14-2
 - introduction to 14-1
 - invisible segments 14-3
 - nondynamic segments 14-3
 - nonretained graphics 14-2
 - on filled primitives 14-2
 - on outlined primitives 14-2
 - parameters
 - input 14-5
 - IMaxDepth 14-8
 - IMaxHits 14-5
 - IType 14-5
 - performed for certain types of functions 14-2
 - pick aperture
 - description of 14-10
 - determining the contents of 14-1
 - positioning 14-2
 - sizing 14-2, 14-11
 - presentation space handle 14-5
 - propagated detectable segments 14-3
 - retained graphics 14-3, 14-4

- correlation (*continued*)
 - segment attribute for 13-8
 - segment ids 14-3
 - segment id—tag pairs 14-4, 14-11
 - segment tags 14-3
 - tagging primitives 14-4
 - tags 14-4
 - types of segments to correlate on 14-5
 - using 14-11
- cosmetic lines 3-3
- Courier font 9-3
- creating
 - bit maps 1-4
 - called segments 12-13
 - chained segments 12-12
 - chained-dynamic segments 13-16
 - custom fill pattern from a bit map 5-16
 - custom fill pattern from a font character 5-19
 - device context 2-9
 - device context for printing 18-19
 - element types 13-5
 - elements 13-4
 - hollow characters 10-10
 - micro presentation spaces, functions 2-2
 - new printer object 18-31
 - new region 11-14
 - new thread 18-18
 - nonretained graphic segments 12-11
 - output page 18-23
 - presentation space 2-6, 18-18
 - print jobs 18-3
 - printer-specific format 18-3
 - regions 11-3, 11-9
 - retained graphics 12-1
 - rubber-banding effect 3-20
 - segment contents 12-4
 - segments 12-3
 - triangular clip path 10-19
- creating a region 11-4
- CREA_DEFAULT 15-15
- CREA_NOREALIZE 15-15
- CRGN_AND 11-5, 16-7
- CRGN_COPY 11-5
- CRGN_DIFF 11-5, 16-7
- CRGN_OR 11-5
- CRGN_XOR 11-5
- CTAB_DEFAULT 15-15
- CTAB_NOMODIFY 15-15
- CTAB_REPLACEPALETTE 15-15
- current
 - attribute modes 12-9, 14-4
 - color 4-4
 - correlation tag 14-4
 - device 4-3
 - drawing mode 13-13
 - element pointer position 14-4
 - marker set 4-2
 - marker symbol 4-1

- current (*continued*)
 - position 4-1, 12-4
 - position, setting 3-2
- current arc 3-10
- current attribute value 1-3
- current clip region 11-7
- current color 5-5
- current font 9-17
- current path 10-8
- current pattern symbol, definition 5-2
- current position 6-8, 6-10
- current position, description 2-5
- current transformation 17-22
- current transformation value 17-22
- currentness, attribute 2-5
- custom fill patterns from a bit map 5-4
- custom fill patterns from a font symbol 5-4
- customizing
 - pattern sets 5-4

D

- dash-dot line 3-4
- dash-double-dot line 3-4
- data structure
 - CHARBUNDLE 9-4
 - FATTRS 9-11
 - FONTMETRICS 9-4
- data structures
 - ARCPARAMS 3-10
 - AREABUNDLE 1-3, 3-10, 5-6, 5-9, 5-13, 5-20, 8-20, 11-3
 - BITMAPHEADER2 8-2
 - BITMAPINFOHEADER2 8-24
 - BITMAPINFO2 8-2, 8-24
 - CHARBUNDLE 1-4, 6-1, 6-2, 6-3, 6-17, 6-23
 - device context 1-2
 - FATTRS 6-1, 6-23
 - FONTMETRICS 6-1, 6-5, 6-23
 - LINEBUNDLE 1-3, 3-2, 5-9, 5-14, 5-20
 - MARKERBUNDLE 1-4, 4-5
 - MATRIXLF 17-13
 - MATRIXLF, and reflecting 17-17
 - MATRIXLF, and rotating 17-19
 - MATRIXLF, and scaling 17-17
 - MATRIXLF, and shearing 17-21
 - MATRIXLF, and translating 17-20
 - MATRIXLF, structure 17-13
 - MATRIXLF, transformation structure 17-4
 - PARCPARAMS 3-10
 - POINTL 5-20, 6-10, 6-23
 - POLYGON 5-13, 5-20
 - presentation space 1-2
 - PRQINFO3 18-14
 - RECTL 11-9
 - RGB 8-7
 - RGB2 8-7
 - summary of transformation structures 17-44, 17-45

data structures (*continued*)

- TGB2 8-24
- data types
- DBM_BACKGROUND 8-9
- DBM_HALFTONE 8-9
- DBM_INVERT 8-9
- DBM_STRETCH 8-9
- DCTL_BOUNDARY 13-15, 16-10
- DCTL_CORRELATE 13-15, 14-2
- DCTL_DISPLAY 13-15, 15-4, 15-8
- DCTL_DYNAMIC 13-13, 13-15
- DCTL_ERASE 13-15
- DCTL_OFF 15-8
- DDEF_DEFAULTS 15-16
- DDEF_IGNORE 15-16
- DDEF_LOADDISC 15-16
- default color table 13-14
- default device transformation 17-34, 17-36
- default logical color table 7-4
- default page coordinate space 1-6
- default pattern set, description 5-4
- default pattern symbol 5-3
- default values, line primitive attributes 3-2
- default values, path attributes 10-2
- default viewing transformation 17-32
- default viewing transformations 17-32
- defining
 - circle 3-14
 - ellipse 3-12
 - lines with geometric width 10-14
 - regions 11-1
 - text appearance 9-1
 - viewing window 17-31
- defining an area primitive, sample code 5-7
- deleting
 - between two element labels 13-11
 - elements 13-11
 - job 18-31
 - open segments 13-11
 - presentation space 2-6
 - presentation spaces 13-11
 - range of elements 13-11
 - region 11-9
 - segments 13-16
- design choices for device fonts 18-17
- designing
- destination region 11-4
- destroying
- detectability attribute of graphics segments 13-8
- detectable primitives 14-2
- determining
 - coordinates of rectangles in region 11-13
 - current position 3-2
 - device capabilities 2-12
 - filled portion of path 10-11
 - region characteristics 11-6
 - what lies in the pick aperture 14-1

- DevCloseDC 1-8, 2-8, 2-12, 15-9, 15-23, 18-25, 18-32
- DevCopyMetaFile 15-23
- DevDeleteMetaFile 15-23
- DevEscape 1-8, 15-6, 18-4, 18-32
- DevEscape device functions 15-4
- DevEscape with DEVEESC_ENDDOC 18-23
- DEVEESC_ABORTDOC. 18-23
- DEVEESC_ENDDOC 18-23
- DEVEESC_GETSCALINGFACTOR 15-3, 15-4
- DEVEESC_NEWFRAME 18-23
- DEVEESC_QUERYESCSUPPORT 15-3, 15-4
- DEVEESC_RAWDATA 18-4
- DEVEESC_STARTDOC 18-22
- device context 15-7
 - about 2-7
 - as a data structure 1-2
 - associating with presentation space 18-21
 - associating with presentation spaces 2-10
 - association with presentation spaces 1-2
 - cached micro 2-7
 - closing 2-12
 - creating 2-9
 - creating for printing 18-19
 - description 1-2, 2-1
 - description of cached 2-7
 - description of normal 2-8
 - description of window 2-7
 - determining device capabilities 2-12
- DevCloseDC 2-8
- DevOpenDC 2-8
- direct 2-8
- disassociating the presentation space 18-25
- information 2-8
- memory 2-8
- metafile 2-8
- Metafile_NoQuery 2-8
- normal 2-7
- OD_INFO 18-17
- OD_MEMORY 18-18
- OD_QUEUED 18-17
- opening 18-18
- opening a queued 18-19
- purpose 1-2
- queued 2-8
- regions 1-5
- summary of functions and structures 2-13
- types 2-7
- using 2-9
- window 2-7
- WinOpenWindowDC 2-7

- device coordinate space 1-6
- device creation, querying, and manipulation. 18-31
- device fonts 9-12
- device fonts, description 18-17
- device functions
 - DevQueryCaps 6-8, 9-11, 9-22
- device independent color indexing 7-5

- device space 17-4
- device space limits 17-36
- device transformation 17-33, 17-34, 17-36
- device transformation matrix 17-37
- device-independence 1-1
- device-independence considerations 18-22
- DevLoadMetaFile 15-23
- DevOpenDC 1-8, 2-8, 2-12, 8-3, 8-15, 15-7, 15-23, 18-32
- DevOpenDC parameters, list 18-19
- DEVOPENSTRUC 2-9, 2-14, 15-8, 18-33
- DevPlayMetaFile 15-23
- DevPostDeviceModes 1-8, 18-15, 18-26, 18-32
- DevQueryCaps 1-8, 2-13, 6-8, 7-17, 18-23, 18-32
- DevQueryDeviceNames 1-8
- DevQueryHardcopyCaps 1-8, 2-13, 18-11, 18-26, 18-32
- DevQueryMetaFileBits 15-23
- DevQueryMetaFileLength 15-23
- DevSaveMetaFile 15-23
- DevSetMetaFileBits 15-23
- direct device context 2-8
- direct manipulation
- direct printing, description 18-7, 18-30
- direction of the arc 3-11
- disassociating a presentation space from a device context 2-10
- disassociating the presentation space 18-25
- disjoint regions, combining 11-5
- display drawing control flag 13-15
- displaying
 - job properties dialog 18-15
- displaying dialogs, printing 18-4
- dithering, definition 7-17
- DM_DRAW 8-9, 8-10, 12-2, 12-11, 13-7, 13-9, 15-3, 15-10
- DM_DRAWANDRETAIN 8-10, 12-2, 13-7, 13-9, 15-3, 15-5, 15-10
- DM_PRINTOBJECT 18-26
- DM_RETAIN 8-10, 12-1, 12-2, 12-12, 12-13, 13-9, 15-3, 15-5, 15-6, 15-10
- DosClose 8-16, 18-9
- DosOpen 8-16, 18-9
- DosWrite 8-16, 18-9
- dotted line 3-4
- double-dotted line 3-4
- DPDM_QUERYJOBPROPS 18-15
- dragging
- DRAGITEM 18-26
- drag/drop protocol considerations 18-26
- draw-and-retain 12-2
- draw-mode restrictions 15-4
- drawing
 - aligning text 6-20
 - box 3-9
 - called segments 12-8
 - chained segments 12-8
 - character string primitives 6-18
 - character strings vertically 6-14
 - characters from left to right 6-12

- drawing (*continued*)
 - circle 3-22
 - circles 3-1
 - clearing the screen beforehand 13-14
 - dynamic segments 13-8, 13-14, 17-29
 - ellipse 3-23
 - ellipses 3-1
 - entire segment chain 13-13
 - filled polygons 10-17
 - fillet 3-25
 - fillets 3-1
 - formatting text on a page 6-19
 - functional sequence for retained graphics 14-5
 - geometric figures 3-1
 - geometric (wide) line 10-16
 - groups of segments 12-8
 - individual segments 12-8
 - letters as a point 6-5
 - letters as a straight line 6-5
 - letters in reverse 6-5
 - multiple arcs 3-17
 - multiple, intersecting closed figures 5-16
 - outline text 10-18
 - partial segment chain 13-13
 - pie slice 3-23
 - polygons 3-1
 - retained graphics 12-1, 12-6
 - root segments 12-8
 - segment chains 12-13
 - segments 13-13
 - single, closed figure 5-15
 - spline 3-26
 - straight line 3-20
 - text 6-18
 - with GpiPolygons 5-13
 - with polygon primitives 5-13
- drawing a closed figure, sample code 5-15
- drawing control
 - boundary flag 13-15
 - correlation flag 13-15, 14-2
 - DCTL_BOUNDARY 13-15
 - DCTL_CORRELATE 13-15, 14-2
 - DCTL_DISPLAY 13-15
 - DCTL_DYNAMIC 13-13, 13-15
 - DCTL_ERASE 13-15
 - display flag 13-15
 - draw-dynamic-segments flag 13-15
 - erase-before-draw flag 13-15
 - introduction to 13-14
 - table of flags 13-15
- drawing mode
 - about 12-2
 - actual 12-6, 13-2
 - affects on dynamic segments 13-7
 - current 13-13
 - definition of 13-2
 - different from actual drawing mode 13-2
 - DM_DRAW 12-2, 13-7, 13-9

drawing mode (*continued*)

- DM_DRAWANDRETAIN 12-2, 13-7, 13-9
- DM_RETAIN 12-1, 12-2, 12-12, 12-13, 13-9
- draw 12-2, 13-2, 14-2
- draw-and-retain 12-2, 12-3, 13-2, 14-2
- effect on deleting elements 13-11
- effects element copying 13-11
- effects on editing segments 13-9
- invalid queries in retain mode 12-2
- retain 12-2, 12-12, 12-13, 13-2
- valid queries in draw-and-retain mode 12-3

drawing modes 15-8

drawing order

- See graphics order

drawing tools, line and arc primitives 3-1

drawing units, setting 17-39

drawing-surface color 6-16

drawing-surface colors 4-4, 5-5, 10-7

DRIVDATA 18-14, 18-33

DRIVPROPS 18-33

dropping

DRO_FILL 3-10, 3-14

DRO_OUTLINE 3-10, 3-14, 3-22

DRO_OUTLINEFILL 3-10, 3-14

dynamic drawing control flag 13-15

dynamic segments 7-15, 13-7, 13-14, 15-6, 16-8

dynamic segments, drawing 17-29

E

edit modes, graphics segments 13-9

editing

mode

- default 13-9
- effect on copying segments 13-12
- SEGEM_INSERT 13-4
- SEGEM_REPLACE 13-4
- restrictions on GpiPutData 13-4
- retained graphics 13-1
- segment contents 13-16
- segments 13-1, 13-9
- specific graphics elements 13-5

Editor

- Font 9-25

effect of programming in pels, example 18-22

element offset 13-12

element pointer 13-5, 13-10

element types 13-4, 13-5

elements

- addressing specific graphics element 13-5
- bracket usage 13-4
- composition of 13-3
- copying 13-11, 13-12
- correlation 14-4
- correlation tags 14-4
- creating 13-4
- creating types 13-5
- definition of 13-1

elements (*continued*)

- deleting 13-11
- deleting between two labels 13-11
- deleting range of 13-11
- editing 13-9
- editing specific graphics element 13-5
- element pointer position 14-4
- inserting 13-9, 13-10
- labeling 13-5
- label—element pair 13-5
- locating specific 13-5
- moving pointer 13-5
- offset 13-12
- pointer 13-5, 13-10
- replacing 13-10
- SEGEM_INSERT 13-10
- SEGEM_REPLACE 13-10
- single pointer position 13-4
- types 13-4, 13-5
- valid construction 13-4
- zero position 13-5

elements of a graphics segment

- functions valid in B-1

ellipses 3-10

ellipse, drawing 3-23

em height 6-22

ending a print job 18-23

ending print cleanup 18-25

equations, transformation 17-11

erase-before-drawing drawing control flag 13-15

error, round-off 17-27

establishing

- segment priority 12-7

examining

example of defining a clipping region 16-9

examples

- a bit map and its associated color table 8-8
- a hexagon and a circle 16-8
- adding a rectangular area to a clip region 16-13
- alternate and winding fill modes 10-11
- application interface and data flow 18-7
- application page setup dialog 18-10
- application printer setup dialog 18-13, 18-18
- area 5-1
- bar graph drawn with line and arc primitives 3-1
- bit map shown on two types of displays 8-2
- bit map transfer 8-10
- bit-map drawing functions 8-8
- bits and pels in a bit-map image 8-1
- blueprint created with line and arc primitives 3-1
- calculating filled areas of alternate mode 5-10
- calculating filled paths constructed in alternate mode 10-11
- character clipping results for device and system fonts 18-17
- closed figure bounded by chord and arc 3-24
- closing unattached geometric lines 10-5
- combining overlapping regions 11-5

examples (continued)

- copying a display-screen image to a bit map 8-18
- copying a graphic segment into a metafile 15-19
- copying a metafile to disk 15-20
- creating a clip path 16-11
- creating a clip region 16-12
- creating a custom fill pattern 8-21
- creating a logical color table 7-20
- creating a metafile 15-18
- creating a non-display device context for a printer 2-9
- creating a palette 7-23
- creating and displaying bit maps 8-5
- creating standard device context for metafile 2-9
- defining a clipping region 16-9
- defining a region 11-2
- defining lines with geometric width 10-14
- determining color-table format and index values 7-21
- determining the index value of an RGB value 7-22
- determining the size of a clipping area 16-14
- DEVOPENSTRUC 2-9
- drawing a box 3-9
- drawing techniques with paths 10-1
- effect of programming in pels 18-22
- ellipse 3-12
- example of setting primitive color attributes 7-22
- excluding a rectangular area from a clip region 16-12
- exclusive-OR (XOR) mix attribute 7-16
- fillet 3-17
- flow for device escapes 18-23
- flow of application graphics commands 1-2
- full arc 3-14
- geometric lines and normal lines 10-4
- GpiLine to draw a single point 3-7
- GpiLine—the current position 3-7
- GpiPolyLine drawing sequence of connected lines 3-8
- GpiPolyLineDisjoint, graph with discontinuities 3-8
- graphics primitives 1-3
- healing screen with clip regions 11-8
- image-primitive bits and pels 8-10
- inclusive/inclusive and inclusive/exclusive clipping 16-3
- joining wide lines 10-5
- line drawn with the GpiLine function 2-4
- loading a bit map from a file 8-22
- metafile operations 15-7
- OR mix attribute 7-14
- overpaint background mix attributes 7-14
- overpaint foreground mix attributes 7-14
- overview of the application interface and data flow 18-6
- partial arc 3-16
- path construction in different directions in winding mode 10-12
- paths constructed in the same direction in winding mode 10-12

examples (continued)

- pattern reference point 5-3
- pie chart drawn with line and arc primitives 3-1
- playing a metafile 15-21
- playing another metafile 15-21, 15-22
- points count for bit-block transfers 8-12
- quarter-circle box-corner rounding 3-9
- relationship between queues and devices 18-5
- scaling and drawing a bit-map image 8-20
- screen repairing 16-7
- setting foreground color attributes 7-23
- setting primitive color attributes 7-22
- setting the clip region to a region intersection 16-14
- specifying a file name for pszLogAddress 18-25
- spline 3-19
- spline with no discontinuity of gradient 3-19
- storing a bit map in a metafile 8-22
- the background of a primitive 7-10
- the bounding rectangle 16-10
- the box 5-11
- the clipping path 16-4
- the foreground of a primitive 7-9
- the graphics field 16-5
- the logical color table 7-3
- the viewing window 16-5
- tilted ellipse 3-14
- triangular clip path 10-15, 16-1
- using the ARE and FIT parameters 18-29
- 3-point arc 3-16
- exchanging
- exclusive OR mode
 - See XOR mode
- exclusive-OR mix 7-15
- executing
- existing transform 17-27
- exporting
- extra space for characters 6-15

F

- face name of outline font 9-5
- family name of outline font 9-5
- FASTATTR_CHAINED 12-4, 12-5, 12-11
- FATTRS 6-23
- FATTRS (font attributes) 9-22
- file system, printing 18-1
- files
- fill modes, path 10-11
- fill path 10-9
- fillet sharpness values 3-18
- fillets 3-10, 3-17
- fillet, drawing 3-25
- finding
 - specific graphics element 13-5
- FIT parameter 18-27
- fixed-point values 6-7

flags

- DCTL_BOUNDARY 16-10
- DCTL_DISPLAY 15-4
- for boundary control 13-15
- for correlation 13-15, 14-2
- for display control 13-15
- for dynamic-segments drawing 13-15
- for erase-before-draw control 13-15
- LCOL_OVERRIDE_DEFAULT_COLORS 7-8
- LCOL_PURECOLOR 7-8

flow for device escapes 18-23

FM_AND 7-12

FM_DEFAULT 7-12, 15-5

FM_INVERT 7-12

FM_LEAVEALONE 7-12, 15-5

FM_MASKSRCNOT 7-12

FM_MERGENOTSRC 7-13

FM_MERGESRCNOT 7-12

FM_NOTCOPYSRC 7-13

FM_NOTMASKSRC 7-13

FM_NOTMERGESRC 7-12

FM_NOTXORSRC 7-12

FM_ONE 7-13

FM_OR 7-12, 15-5

FM_OVERPAINT 3-5, 4-5, 5-5, 7-12, 15-5

FM_SUBTRACT 7-12

FM_XOR 7-12

FM_ZERO 7-12

font attributes (FATTRS) 9-22

font dialog, description 18-9

Font Editor 9-25

font family

- definition

font metrics 9-18

font-use indicators 9-13

FONTMETRICS 6-23

fonts

See also character string primitives

about 9-1

Adobe fonts 9-4

affected by character attributes 6-3

appearance 6-1

- condensed 9-1

- expanded 9-1

- italic 9-1

attributes

baselines 6-8, 6-9, 6-12

creating 9-25

creating custom fill pattern from character 5-19

current 9-17

custom fill patterns from a symbol 5-4

definition of 6-1

description 1-4

device 9-12, 18-17

em height 6-22

family

- definition 9-1

generic 9-12

fonts (continued)

how image fonts obey character attributes 6-4

how outline fonts obey character attributes 6-4

image 6-1, 6-4, 6-6, 6-9, 6-10, 6-16, 9-2

Kerning 6-15

lcid identified resource 9-1

line weight 6-1

- bold 9-1

- normal 9-1

logical 9-17

- defining 9-22

match value 9-12

metrics 6-7

monospace 6-4

outline 6-1, 6-4, 6-8, 6-10, 6-16, 9-2

physical 9-17

point size 6-1, 6-4, 6-8, 6-19, 6-22

private 9-14

proportional 6-4

proportional spacing 6-15

public 9-14

raster 9-2

selecting 9-17

serif 6-1

- definition 9-1

stroke width 6-1

system 9-2, 18-17

transformable 9-2

types 18-17

used with character string primitives 9-1

using arc primitives 6-2

using bit maps 6-2

using line primitives 6-2

vector 9-2

foreground color 5-1, 6-16, 7-9

foreground colors 4-4, 5-5

foreground mix 4-4, 5-5, 7-11

- background mix 7-11

foreground mix attribute 5-1, 6-16, 12-8

foreground mix, marker 4-4

formatting

- page 18-18

forms selection 18-11

FPATH_ALTERNATE 10-11, 10-16

FPATH_WINDING 10-11, 10-16

freeing

full arcs 3-12

functional sequence for drawing outlines 10-10

functional sequence, list 5-12

functions

- categories for spooler management 18-31

- clipping 10-15

- DevCloseDC 2-8, 2-12, 15-9, 15-23, 18-25, 18-32

- DevCopyMetaFile 15-23

- DevDeleteMetaFile 15-23

- DevEscape 15-6, 18-4, 18-22, 18-32

- DEVESC_GETSCALINGFACTOR 15-3, 15-4

- DEVESC_QUERYYESCSUPPORT 15-3, 15-4

functions (*continued*)

DevLoadMetaFile 15-23
 DevOpenDC 2-8, 2-12, 8-3, 8-15, 15-23, 18-19, 18-32
 DevPlayMetaFile 15-23
 DevPostDeviceModes 18-15, 18-26, 18-32
 DevQueryCaps 2-13, 6-8, 7-17, 18-23, 18-32
 DevQueryHardcopyCaps 2-13, 18-11, 18-26, 18-32
 DevQueryMetaFileBits 15-23
 DevQueryMetaFileLength 15-23
 DevSaveMetaFile 15-23
 DevSetMetaFileBits 15-23
 DosClose 8-16
 DosOpen 8-16
 DosWrite 8-16
 GpiAnimatePalette 7-24
 GpiAssociate 2-13, 12-11, 15-8, 18-21
 GpiBeginArea 3-23, 5-7, 5-12, 5-20, 14-4
 GpiBeginElement 13-1, 13-4, 13-5, 13-18
 GpiBeginPath 10-8, 16-3, 16-11
 GpiBitBlt 8-9, 8-10, 8-15, 15-4, 15-6
 GpiBox 3-6, 3-10, 5-11, 14-2, 17-28
 GpiCallSegMatrix 17-11
 GpiCallSegmentMatrix 12-5, 12-12, 12-14, 17-24, 17-27, 17-29, 17-30, 17-43
 GpiCharString 6-13, 6-19, 6-21, 6-23
 GpiCharStringAt 6-13, 6-19, 6-21, 6-23
 GpiCharStringPos 6-13, 6-19, 6-21, 6-23, 18-17
 GpiCharStringPosAt 6-13, 6-19, 6-20, 6-21, 6-23
 GpiCloseFigure 10-8
 GpiCloseSegment 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-9, 13-11, 13-16, 13-18
 GpiColor 5-12
 GpiCombineRegion 11-3, 11-10, 16-7, 16-13
 GpiConvert 6-20, 17-36, 17-38
 GpiCopyMetaFile 15-16
 GpiCorrelateChain 14-4, 14-5, 14-11, 14-12
 GpiCorrelateFrom 14-4, 14-12
 GpiCorrelateSegment 14-4, 14-12
 GpiCreateBitmap 8-2, 8-4, 8-6, 8-24
 GpiCreateLogColorTable 7-5, 7-6, 7-17, 7-24
 GpiCreateLogFont 18-17
 GpiCreateLogicalFont 6-3
 GpiCreatePalette 7-24
 GpiCreatePS 2-11, 2-13, 12-11, 15-8, 17-2, 17-3, 17-4, 17-6, 17-7, 17-34, 17-39
 GpiCreateRegion 11-3, 11-9, 16-12, 16-13
 GpiDeleteBitmap 8-17
 GpiDeleteElement 13-11, 13-16, 13-18
 GpiDeleteElementRange 13-11, 13-16, 13-18
 GpiDeleteElementsBetweenLabels 13-11, 13-18
 GpiDeleteMetaFile 15-17
 GpiDeletePalette 7-8, 7-24
 GpiDeleteSegment 13-11, 13-18
 GpiDeleteSegments 13-11, 13-18
 GpiDeleteSement 13-11
 GpiDestroy 12-10
 GpiDestroyPS 2-6, 2-13
 GpiDestroyRegion 11-9

functions (*continued*)

GpiDrawBits 8-9, 8-10, 8-24
 GpiDrawBlt 8-10
 GpiDrawChain 11-8, 12-5, 12-6, 12-7, 12-8, 12-13, 12-14, 13-7, 13-13, 13-18, 17-28
 GpiDrawDynamics 13-14, 13-16, 13-18, 15-6, 17-29
 GpiDrawFrom 12-8, 12-14, 13-7, 13-13, 13-18, 17-28
 GpiDrawSegment 12-8, 12-14, 13-7, 13-13, 13-15, 13-18, 17-28
 GpiElement 13-4, 13-5, 13-11, 13-14, 13-18, 14-2
 GpiEndArea 3-23, 5-7, 5-12, 5-20, 14-4
 GpiEndElement 13-1, 13-4, 13-18
 GpiEndPath 10-8, 16-3, 16-11
 GpiEqualRegion 11-3, 11-6, 11-11
 GpiErase 13-14, 13-18, 14-2, 14-12, 15-4, 15-6
 GpiErrorSegmentData 13-13, 13-18
 GpiExcludeClipRectangle 11-8, 15-4, 15-5, 16-7, 16-12, 16-15
 GpiExcludeClipRegion 15-4
 GpiFillPath 10-11
 GpiFrameRegion 11-2, 11-12
 GpiFullArc 3-22
 GpiGetData 13-4, 13-12, 13-18
 GpiImage 8-9, 8-24
 GpiIntersectClipRectangle 11-8, 15-4, 15-5, 16-7, 16-15
 GpiIntersectClipRegion 15-4
 GpiLabel 13-6, 13-18
 GpiLine 2-4, 3-6, 5-12, 13-2, 13-4, 17-2
 GpiLoadBitmap 8-5, 8-24
 GpiLoadMetaFile 15-16
 GpiMarker 4-1
 GpiModifyPath 10-2, 10-6, 10-13, 16-3
 GpiMove 2-5, 2-13, 3-2, 3-17, 5-7, 12-9
 GpiOffsetClipRegion 11-8, 15-4, 15-5, 16-7, 16-15
 GpiOffsetElementPointer 13-5, 13-6, 13-16, 13-18
 GpiOffsetRegion 11-6, 11-11
 GpiOpenSegment 12-3, 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-5, 13-6, 13-9, 13-10, 13-16, 13-18, 14-3, 17-27
 GpiOutlinePath 10-6, 10-18
 GpiPaintRegion 11-2, 11-12, 15-4, 15-6
 GpiPartialArc 3-14, 3-23
 GpiPathToRegion 10-16
 GpiPlayMetaFile 14-2, 15-10, 15-13, 17-43, 18-4
 GpiPlaySegment 17-11
 GpiPolyFillet 3-17
 GpiPolyFilletSharp 3-17
 GpiPolygons 5-13, 5-15, 5-20
 GpiPolyLine 3-6, 5-12
 GpiPolyLineDisjoint 3-2, 3-6
 GpiPolyMarker 4-1
 GpiPolySpline 3-17, 3-19
 GpiPop 12-9, 14-4
 GpiPtInRegion 11-12
 GpiPtVisible 11-8, 16-8, 16-15
 GpiPutData 13-4, 13-10, 13-12, 13-14, 13-18, 14-2, 17-43

functions (continued)

GpiQueryAttrMode 2-5
 GpiQueryAttrs 2-5, 3-5, 4-4, 5-5, 6-17, 8-9
 GpiQueryBackColor 4-4, 6-17, 7-24
 GpiQueryBackMix 4-4, 6-17, 7-17, 7-24
 GpiQueryBitmapBits 8-16, 8-24
 GpiQueryBitmapDimension 8-24
 GpiQueryBitmapHandle 8-24
 GpiQueryBitmapInfoHeader 8-24
 GpiQueryBitmapParameters 8-16, 8-24
 GpiQueryBoundaryData 14-12, 16-10
 GpiQueryCharAngle 6-8, 6-23
 GpiQueryCharBox 6-7, 6-23
 GpiQueryCharBreakExtra 6-15, 6-23
 GpiQueryCharDirection 6-13, 6-23
 GpiQueryCharExtra 6-15, 6-23
 GpiQueryCharMode 6-23
 GpiQueryCharSet 6-3, 6-23
 GpiQueryCharShear 6-10, 6-23
 GpiQueryCharStringPos 6-20, 6-21, 6-23, 18-17
 GpiQueryCharStringPosAt 6-20, 6-21, 6-23
 GpiQueryClipBox 11-8, 16-7, 16-15
 GpiQueryClipRegion 16-15
 GpiQueryColor 4-4, 6-17, 7-24
 GpiQueryColorData 7-7, 7-24
 GpiQueryColorIndex 7-7, 7-25
 GpiQueryCurrentPosition 3-2, 6-20
 GpiQueryDefAttrs 6-7
 GpiQueryDefaultViewMatrix 17-32
 GpiQueryDefCharBox 6-23
 GpiQueryDeviceBitmapFormats 8-4, 8-7, 8-24
 GpiQueryDrawControl 13-18
 GpiQueryDrawingMode 12-2, 13-18
 GpiQueryEditMode 13-9, 13-18
 GpiQueryElement 13-5, 13-6, 13-11, 13-18
 GpiQueryElementPointer 13-5, 13-19
 GpiQueryElementType 13-4, 13-19
 GpiQueryFonts 18-17
 GpiQueryGraphicsField 6-14, 6-20, 16-15
 GpiQueryInitialSegmentAttrs 12-4, 12-14, 13-7, 13-19
 GpiQueryLineJoin 10-5
 GpiQueryLineWidthGeom 3-3
 GpiQueryLogColorTable 7-7, 7-25
 GpiQueryMix 4-4, 6-17, 7-16, 7-25
 GpiQueryModelTransformMatrix 17-28
 GpiQueryNearestColor 7-7, 7-18, 7-25
 GpiQueryPageViewport 6-14, 6-20, 17-34, 17-36
 GpiQueryPalette 7-8, 7-25
 GpiQueryPaletteInfo 7-25
 GpiQueryPattern 5-2
 GpiQueryPatternRefPoint 5-3, 5-20
 GpiQueryPatternSet 5-20
 GpiQueryPattn 5-20
 GpiQueryPel 8-15, 8-24
 GpiQueryPickAperturePosition 14-10, 14-12
 GpiQueryPickApertureSize 14-10, 14-12
 GpiQueryPS 2-13, 17-34

functions (continued)

GpiQueryRealColors 7-7, 7-25
 GpiQueryRegionRects 11-3, 11-7, 11-13
 GpiQueryRGBColor 7-7, 7-25
 GpiQuerySegmentAttrs 12-14, 13-9, 13-19
 GpiQuerySegmentNames 12-3, 12-14, 13-19
 GpiQuerySegmentPriority 12-7, 12-14, 13-19
 GpiQuerySegmentTransformMatrix 17-29
 GpiQueryStopDraw 13-19
 GpiQueryTag 14-4, 14-12
 GpiQueryTextBox 6-20, 6-23
 GpiQueryViewingLimit 6-14, 6-20
 GpiQueryViewingLimits 16-15
 GpiQueryViewingTransformMatrix 17-32
 GpiRealizeColorTable 7-25
 GpiRectVisible 11-8, 16-8, 16-15
 GpiRemoveDynamics 13-14, 13-16, 13-19, 15-6, 17-29
 GpiResetBoundaryData 14-12, 16-10
 GpiResetPS 2-13, 12-10, 12-11, 13-11, 15-6, 15-12
 GpiRestorePS 2-6, 2-13, 12-10
 GpiRotate 17-14, 17-17, 17-26, 17-39
 GpiSaveMetaFile 15-16
 GpiSavePS 2-6, 2-13, 6-7, 12-10, 15-13
 GpiScale 17-14, 17-15, 17-26, 17-27, 17-39
 GpiSelectPalette 7-25
 GpiSetArcParams 3-10
 GpiSetAttrMode 2-5, 12-9
 GpiSetAttrs 2-5, 3-5, 4-4, 5-5, 6-3, 6-17, 7-6, 7-10, 7-17, 8-9
 GpiSetBackColor 4-4, 5-5, 6-17, 7-25
 GpiSetBackMix 4-4, 5-5, 6-17, 7-17, 7-25
 GpiSetBitmap 8-4, 8-24
 GpiSetBitmapBits 8-16, 8-24
 GpiSetBitmapDimension 8-24
 GpiSetBitmapId 8-6, 8-24
 GpiSetCharAngle 6-3, 6-8, 6-9, 6-23
 GpiSetCharBox 6-3, 6-5, 6-7, 6-23
 GpiSetCharBreakExtra 6-3, 6-15, 6-23
 GpiSetCharDirection 6-3, 6-12, 6-23
 GpiSetCharExtra 6-3, 6-15, 6-23
 GpiSetCharMode 6-3, 6-23
 GpiSetCharSet 6-3, 6-23
 GpiSetCharShear 6-3, 6-10, 6-11, 6-23
 GpiSetClipPath 10-15, 16-3, 16-4, 16-11, 16-15
 GpiSetClipRegion 11-7, 11-9, 15-4, 15-5, 16-6, 16-12, 16-13, 16-15
 GpiSetColor 4-4, 5-5, 6-17, 7-6, 7-10, 7-25
 GpiSetCurrentPosition 2-5, 2-13, 3-2, 5-7, 6-21, 12-4, 12-9
 GpiSetDefAttrs 4-4, 5-4, 6-7, 6-17, 12-4
 GpiSetDefaultViewMatrix 17-11, 17-30, 17-32
 GpiSetDefTag 14-4
 GpiSetDrawControl 13-14, 13-15, 13-19, 14-2, 14-12, 15-4, 15-8, 16-10
 GpiSetDrawingMode 12-2, 12-6, 12-14, 13-2, 13-19
 GpiSetEditMode 13-9, 13-16, 13-19
 GpiSetElementPointer 13-5, 13-6, 13-16, 13-19

functions (*continued*)

- GpiSetElementPointerAtLabel 13-6, 13-16, 13-19
- GpiSetGraphicsField 16-5, 16-15, 17-32
- GpiSetInitialSegmentAttrs 12-4, 12-5, 12-12, 12-13, 12-14, 13-7, 13-9, 13-19
- GpiSetLineEnd 10-4
- GpiSetLineJoin 10-5
- GpiSetLineType 3-23
- GpiSetLineWidth 3-3
- GpiSetLineWidthGeom 3-3, 10-2
- GpiSetMix 4-4, 5-5, 6-17, 7-16, 7-25, 13-8
- GpiSetModelTransformMatrix 17-27, 17-28, 17-30
- GpiSetPageViewport 17-3, 17-11, 17-34, 17-35
- GpiSetPaletteEntries 7-8, 7-25
- GpiSetPattern 5-2, 5-20
- GpiSetPatternRefPoint 5-3, 5-4, 5-20
- GpiSetPatternSet 5-20
- GpiSetPel 8-15, 8-24, 15-4, 15-6
- GpiSetPickAperture 14-11
- GpiSetPickAperturePosition 14-2, 14-4, 14-12
- GpiSetPickApertureSize 14-2, 14-4, 14-5, 14-12
- GpiSetPS 2-13, 12-11, 13-11, 15-6, 15-12
- GpiSetPS. 12-10
- GpiSetRegion 11-5
- GpiSetSegmentAttrs 12-14, 13-9, 13-19, 14-3
- GpiSetSegmentPriority 12-3, 12-7, 12-14, 13-9, 13-19
- GpiSetSegmentTransformationsMatrix 17-27, 17-30
- GpiSetSegmentTransformMatrix 17-29
- GpiSetStopDraw 13-19, 15-6
- GpiSetTag 14-3, 14-4, 14-12
- GpiSetTextAlignment 6-3, 6-13, 6-14
- GpiSetViewingLimits 16-5, 16-15, 17-31, 17-32
- GpiSetViewingTransformMatrix 17-30, 17-32, 17-42
- GpiStrokePath 10-2, 10-6, 10-14
- GpiTranslate 17-19, 17-26, 17-39
- GpiUnrealizeColorTable 7-25
- GpiWCBitBlt 8-9, 8-10, 8-11, 8-15, 17-37
- GpuBitBlt 8-10
- line and arc, list 5-8
- no return with disabled spooler, list 18-31
- PrfQueryProfileString 18-14
- query, list 5-9
- recording 15-8
- region, using 11-9
- return values 14-2
- ROP_NOTSRCCOPY 8-10
- sequence for drawing outline 10-10
- specification, list 5-8
- SplControlDevice 18-32
- SplCopyJob 18-32
- SplCreateDevice 18-31, 18-32
- SplCreateQueue 18-31, 18-32
- SplDeleteDevice 18-32
- SplDeleteJob 18-32
- SplDeleteQueue 18-32
- SplEnumDevice 18-32
- SplEnumDriver 18-32

functions (*continued*)

- SplEnumJob 18-32
- SplEnumPort 18-32
- SplEnumPrinter 18-32
- SplEnumQueue 18-13, 18-32
- SplEnumQueueProcessor 18-32
- SplHoldJob 18-32
- SplHoldQueue 18-32
- SplPurgeQueue 18-32
- SplQmAbort 18-32
- SplQmAbortDoc 18-32
- SplQmClose 18-32
- SplQmEndDoc 18-32
- SplQmOpen 18-32
- SplQmStartDoc 18-32
- SplQmWrite 18-32
- SplQueryDevice 18-32
- SplQueryJob 18-32
- SplQueryQueue 18-14, 18-32
- SplReleaseJob 18-32
- SplReleaseQueue 18-32
- SplSetDevice 18-32
- SplSetJob 18-32
- SplSetQueue 18-32
- summary of line and arc 3-27
- summary of transformation functions 17-44
- that generate a path 10-9
- to use with current clip region 11-8
- used in a path 10-9
- WinBeginPaint 2-2, 2-3, 2-4, 2-6, 2-13
- WinDrawBitmap 8-9, 8-24
- WinEndPaint 2-6, 2-13
- WinGetPS 2-3, 2-6, 2-13
- WinGetScreenPS 2-2, 2-3, 2-13
- WinOpenWindowDC 2-7, 2-11, 2-13
- WinQuerySysColor 7-9
- WinQuerySysValue 2-13
- WinQueryUpdateRegion 11-8
- WinQueryWindowDC 2-11, 2-13
- WinRealizePalette 7-8
- WinReleasePS 2-6, 2-13
- functions that draw simple arcs 3-12
- functions, transform 17-14

G

- generic fonts 9-12
- geometric lines 3-3
- geometric lines and normal lines, example 10-4
- geometric width for paths 10-2
- geometric width, line 3-2
- getting
- glyphs 9-8
- GPI
 - See Graphics Programming Interface (GPI)
- GpiAnimatePalette 7-24
- GpiAssociate 2-13, 12-11, 15-8, 18-21, 18-25

GpiBeginArea 1-8, 3-23, 5-7, 5-12, 5-20, 14-4
 GpiBeginElement 13-1, 13-4, 13-5, 13-18
 GpiBeginPath 1-8, 10-8, 10-10, 16-3, 16-11
 GpiBitBlt 1-8, 8-9, 8-10, 8-15, 15-4, 15-6
 GpiBox 3-6, 3-26, 5-11, 14-2, 17-28
 GpiBox primitive 3-9
 GpiCallSegMatrix 17-11
 GpiCallSegmentMatrix 12-5, 12-12, 12-14, 17-24, 17-27, 17-29, 17-30, 17-43
 GpiCharString 1-8, 6-13, 6-19, 6-21, 6-23, 10-10
 GpiCharStringAt 6-13, 6-19, 6-21, 6-23
 GpiCharStringPos 6-13, 6-19, 6-21, 6-23, 18-17
 GpiCharStringPosAt 6-13, 6-19, 6-20, 6-21, 6-23
 GpiCloseFigure 10-8
 GpiCloseSegment 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-9, 13-11, 13-16, 13-18
 GpiColor 5-12
 GpiCombineRegion 1-8, 11-3, 16-7, 16-13
 GpiConvert 1-8, 6-20, 17-36, 17-38
 GpiCopyMetaFile 15-16
 GpiCorrelateChain 14-4, 14-5, 14-11, 14-12
 GpiCorrelateFrom 14-4, 14-12
 GpiCorrelateSegment 14-4, 14-12
 GpiCreateBitmap 1-8, 8-2, 8-4, 8-6, 8-24
 GpiCreateLogColorTable 7-5, 7-6, 7-17, 7-24
 GpiCreateLogFont 1-8, 18-17
 GpiCreateLogicalFont 6-3
 GpiCreatePalette 7-24
 GpiCreatePS 1-8, 2-11, 2-13, 12-11, 15-8, 17-2, 17-3, 17-4, 17-6, 17-7, 17-34, 17-39
 GpiCreateRegion 1-8, 11-3, 11-4, 11-9, 16-12, 16-13
 GpiDeleteBitmap 1-8, 8-17
 GpiDeleteElement 13-11, 13-16, 13-18
 GpiDeleteElementRange 13-11, 13-16, 13-18
 GpiDeleteElementsBetweenLabels 13-11, 13-18
 GpiDeleteMetaFile 15-17
 GpiDeletePalette 7-8, 7-24
 GpiDeleteSegment 13-11, 13-18
 GpiDeleteSegments 13-11, 13-18
 GpiDeleteSement 13-11
 GpiDeleteSetId 1-8
 GpiDestroy 12-10
 GpiDestroyPS 1-8, 2-6, 2-13, 15-9
 GpiDestroyRegion 11-9
 GpiDrawBits 8-9, 8-10, 8-24
 GpiDrawBlt 8-10
 GpiDrawChain 11-8, 12-5, 12-6, 12-7, 12-8, 12-13, 12-14, 13-13, 13-18, 15-8, 17-28
 GpiDrawDynamics 13-14, 13-16, 13-18, 15-6, 17-29
 GpiDrawFrom 12-8, 12-14, 13-13, 13-18, 15-8, 17-28
 GpiDrawSegment 12-8, 12-14, 13-13, 13-15, 13-18, 15-8, 17-28
 GpiElement 13-4, 13-5, 13-11, 13-18, 14-2
 GpiEndArea 1-8, 3-23, 5-7, 5-12, 5-20, 14-4
 GpiEndElement 13-1, 13-4, 13-18
 GpiEndPath 1-8, 10-8, 10-10, 16-3, 16-11
 GpiEqualRegion 11-3, 11-6, 11-11
 GpiErase 1-8, 7-18, 13-14, 13-18, 14-2, 14-12, 15-4, 15-6
 GpiErrorSegmentData 13-13, 13-18
 GpiExcludeClipRectangle 11-8, 15-4, 15-5, 16-7, 16-12, 16-15
 GpiExcludeClipRegion 15-4
 GPIE_DATA 13-14
 GPIE_ELEMENT 13-14
 GPIE_SEGMENT 13-14
 GpiFillPath 10-11
 GpiFrameRegion 11-2, 11-12
 GpiFullArc 3-26, 10-10
 GpiFullArc input parameters 3-14
 GpiGetData 13-4, 13-12, 13-18
 GpiImage 8-9, 8-24
 GpiIntersectClipRectangle 1-8, 11-8, 15-4, 15-5, 16-7, 16-15
 GpiIntersectClipRegion 15-4
 GpiLabel 13-6, 13-18
 GpiLine 1-8, 2-4, 3-6, 3-26, 5-12, 13-2, 13-4, 17-2
 GpiLine to draw a single point 3-7
 GpiLoadBitmap 8-5, 8-24
 GpiLoadFonts 1-8
 GpiLoadMetaFile 15-16
 GpiMarker 4-1
 GpiModifyPath 10-2, 10-6, 10-13, 16-3
 GpiMove 1-8, 2-5, 2-13, 3-2, 3-17, 3-26, 5-7, 10-10, 12-9
 GpiOffsetClipRegion 11-8, 15-4, 15-5, 16-7, 16-15
 GpiOffsetElementPointer 13-5, 13-6, 13-16, 13-18
 GpiOffsetRegion 11-6, 11-11
 GpiOpenSegment 12-3, 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-5, 13-6, 13-9, 13-10, 13-16, 13-18, 14-3, 17-27
 GpiOutlinePath 10-6, 10-10, 10-18
 GpiPaintRegion 1-8, 11-2, 11-12, 15-4, 15-6
 GpiPartialArc 3-14, 3-23, 3-26
 GpiPathToRegion 10-16
 GpiPlayMetaFile 14-2, 15-10, 15-13, 17-43, 18-4
 GpiPlaySegment 17-11
 GpiPointArc 3-26
 GpiPolyFillet 3-17, 3-26
 GpiPolyFilletSharp 3-17, 3-26
 GpiPolygons 1-3, 5-13, 5-15, 5-20
 GpiPolyLine 1-8, 3-6, 3-26, 5-12
 GpiPolyLineDisjoint 3-2, 3-6
 GpiPolyLineDisjoint—graph with discontinuities 3-8
 GpiPolyLine—drawing sequence of connected lines 3-8
 GpiPolyMarker 4-1
 GpiPolySpline 1-8, 3-17, 3-19, 3-26
 GpiPop 12-9, 14-4
 GpiPtInRegion 11-12
 GpiPtVisible 11-8, 16-8, 16-15
 GpiPutData 13-4, 13-10, 13-12, 13-18, 14-2, 17-43
 GpiQueryArcParams 3-26
 GpiQueryAttrMode 2-5
 GpiQueryAttrs 1-8, 2-5, 3-5, 3-26, 4-4, 5-5, 6-17, 8-9

GpiQueryBackColor 4-4, 6-17, 7-24
 GpiQueryBackMix 4-4, 6-17, 7-17, 7-24
 GpiQueryBitmap 1-8
 GpiQueryBitmapBits 1-8, 8-16, 8-24
 GpiQueryBitmapDimension 8-24
 GpiQueryBitmapHandle 1-8, 8-24
 GpiQueryBitmapInfoHeader 8-24
 GpiQueryBitmapParameters 1-8, 8-16, 8-24
 GpiQueryBoundaryData 14-12, 16-10
 GpiQueryCharAngle 6-8, 6-23
 GpiQueryCharBox 6-7, 6-23
 GpiQueryCharBreakExtra 6-15, 6-23
 GpiQueryCharDirection 6-13, 6-23
 GpiQueryCharExtra 6-15, 6-23
 GpiQueryCharMode 6-23
 GpiQueryCharSet 1-8, 6-3, 6-23
 GpiQueryCharShear 6-10, 6-23
 GpiQueryCharStringPos 1-8, 6-20, 6-23, 18-17
 GpiQueryCharStringPosAt 6-20, 6-23
 GpiQueryClipBox 1-8, 11-8, 16-7, 16-15
 GpiQueryClipRegion 16-15
 GpiQueryColor 4-4, 6-17, 7-24
 GpiQueryColorData 7-7, 7-24
 GpiQueryColorIndex 7-7, 7-25
 GpiQueryCurrentPosition 1-8, 3-2, 3-26
 GpiQueryDefAttrs 6-7
 GpiQueryDefaultViewMatrix 17-32
 GpiQueryDefCharBox 6-23
 GpiQueryDevice 1-8
 GpiQueryDeviceBitmapFormats 1-8, 8-4, 8-7, 8-24
 GpiQueryDrawControl 13-18
 GpiQueryDrawingMode 12-2, 13-18
 GpiQueryEditMode 13-9, 13-18
 GpiQueryElement 13-5, 13-6, 13-11, 13-18
 GpiQueryElementPointer 13-5, 13-19
 GpiQueryElementType 13-4, 13-19
 GpiQueryFontMetrics 1-8
 GpiQueryFonts 1-8, 18-17
 GpiQueryGraphicsField 6-14, 6-20, 16-15
 GpiQueryInitialSegmentAttrs 12-4, 12-14, 13-7, 13-19
 GpiQueryLineJoin 10-5
 GpiQueryLineType 1-8, 3-27
 GpiQueryLineWidth 1-8, 3-27
 GpiQueryLineWidthGeom 3-3
 GpiQueryLogColorTable 7-7, 7-25
 GpiQueryMetaFileBits 15-16
 GpiQueryMetaFileLenth 15-16
 GpiQueryMix 1-8, 4-4, 6-17, 7-16, 7-25
 GpiQueryModelTransformMatrix 1-8, 17-28
 GpiQueryNearestColor 7-7, 7-18, 7-25
 GpiQueryNumberSetIds 1-8
 GpiQueryPageViewport 6-14, 6-20, 17-34, 17-36
 GpiQueryPalette 7-8, 7-25
 GpiQueryPaletteInfo 7-25
 GpiQueryPattern 5-2, 5-20
 GpiQueryPatternRefPoint 5-3, 5-20
 GpiQueryPatternSet 1-8, 5-20
 GpiQueryPel 1-8, 8-15, 8-24
 GpiQueryPickAperturePosition 14-10, 14-12
 GpiQueryPickApertureSize 14-10, 14-12
 GpiQueryPS 2-13, 17-34
 GpiQueryRealColors 7-7, 7-25
 GpiQueryRegionBox 1-8
 GpiQueryRegionRects 11-3, 11-7, 11-13
 GpiQueryRGBColor 7-7, 7-25
 GpiQuerySegmentAttrs 12-14, 13-9, 13-19
 GpiQuerySegmentNames 12-3, 12-14, 13-19
 GpiQuerySegmentPriority 12-7, 12-14, 13-19
 GpiQuerySegmentTransformMatrix 17-29
 GpiQueryStopDraw 13-19
 GpiQueryTag 14-4, 14-12
 GpiQueryTextBox 6-20, 6-23
 GpiQueryViewingLimit 6-14, 6-20
 GpiQueryViewingLimits 16-15
 GpiQueryViewingTransformMatrix 17-32
 GpiQueryWidthTable 1-8
 GpiRealizeColorTable 7-25
 GpiRectVisible 1-8, 11-8, 16-8, 16-15
 GpiRemoveDynamics 13-14, 13-16, 13-19, 15-6, 17-29
 GpiResetBoundaryData 14-12, 16-10
 GpiResetPS 1-8, 2-13, 12-10, 12-11, 13-11, 15-6, 15-12
 GpiRestorePS 1-8, 2-6, 2-13, 12-10
 GpiRotate 17-14, 17-17, 17-26, 17-39
 GpiSaveMetaFile 15-16
 GpiSavePS 1-8, 2-6, 2-13, 6-7, 12-10, 15-13
 GpiScale 17-14, 17-15, 17-26, 17-27, 17-39
 GpiSelectPalette 7-25
 GpiSetArcParams 1-8, 3-10, 3-27
 GpiSetAttrMode 2-5
 GpiSetAttrs 1-8, 2-5, 3-5, 3-27, 4-4, 5-5, 6-3, 6-17, 7-6, 7-10, 7-17, 8-9
 GpiSetBackColor 4-4, 5-5, 6-17, 7-10, 7-25
 GpiSetBackMix 4-4, 5-5, 6-17, 7-17, 7-25
 GpiSetBitmap 1-8, 8-4, 8-24
 GpiSetBitmapBits 8-16, 8-24
 GpiSetBitmapDimension 8-24
 GpiSetBitmapid 1-8, 8-6, 8-24
 GpiSetCharAngle 6-3, 6-8, 6-9, 6-23
 GpiSetCharBox 1-8, 6-3, 6-5, 6-7, 6-23
 GpiSetCharBreakExtra 6-3, 6-15, 6-23
 GpiSetCharDirection 6-3, 6-12, 6-23
 GpiSetCharExtra 6-3, 6-15, 6-23
 GpiSetCharMode 6-3, 6-23
 GpiSetCharSet 6-3, 6-23
 GpiSetCharShear 6-10, 6-11, 6-23
 GpiSetClipPath 10-15, 16-3, 16-4, 16-11, 16-15
 GpiSetClipRegion 1-8, 11-9, 15-4, 15-5, 16-6, 16-12, 16-13, 16-15
 GpiSetColor 3-27, 4-4, 5-5, 6-17, 7-6, 7-10, 7-25
 GpiSetCP 1-8
 GpiSetcurrentPosition 1-8, 2-5, 2-13, 3-2, 3-27, 5-7, 6-21, 12-4, 12-9
 GpiSetDefAttrs 4-4, 5-4, 6-7, 6-17, 12-4
 GpiSetDefaultViewMatrix 17-11, 17-30, 17-32

- GpiSetDefTag 14-4
- GpiSetDrawControl 13-14, 13-15, 13-19, 14-2, 14-12, 15-4, 15-8, 16-10
- GpiSetDrawingMode 12-2, 12-6, 12-14, 13-2, 13-19, 15-8
- GpiSetDrawingMode drawing mode
 - See drawing mode
- GpiSetEditMode 13-9, 13-16, 13-19
- GpiSetElementPointer 13-5, 13-6, 13-16, 13-19
- GpiSetElementPointerAtLabel 13-6, 13-16, 13-19
- GpiSetGraphicsField 16-5, 16-15, 17-32
- GpiSetInitialSegmentAttrs 12-4, 12-5, 12-12, 12-13, 12-14, 13-7, 13-9, 13-19
- GpiSetLineEnd 10-4
- GpiSetLineJoin 10-5
- GpiSetLineType 1-8, 3-23, 3-27
- GpiSetLineWidth 1-8, 3-3, 3-27
- GpiSetLineWidthGeom 3-3, 10-2
- GpiSetMetaFileBits 15-16
- GpiSetMix 1-8, 3-27, 4-4, 5-5, 6-17, 7-16, 7-25, 13-8
- GpiSetModelTransformMatrix 1-8, 17-27, 17-28, 17-30
- GpiSetPageViewport 1-8, 17-3, 17-11, 17-34, 17-35
- GpiSetPaletteEntries 7-8, 7-25
- GpiSetPattern 1-8, 5-2, 5-20
- GpiSetPatternRefPoint 5-3, 5-4, 5-20
- GpiSetPatternSet 5-20
- GpiSetPel 1-8, 8-15, 8-24, 15-4, 15-6
- GpiSetPickAperture 14-11
- GpiSetPickAperturePosition 14-2, 14-4, 14-12
- GpiSetPickApertureSize 14-2, 14-4, 14-5, 14-12
- GpiSetPS 1-8, 2-13, 12-10, 12-11, 13-11, 15-6, 15-12
- GpiSetRegion 1-8, 11-5
- GpiSetSegmentAttrs 12-14, 13-9, 13-19, 14-3
- GpiSetSegmentPriority 12-3, 12-7, 12-14, 13-9, 13-19
- GpiSetSegmentTransformationsMatrix 17-27, 17-30
- GpiSetSegmentTransformMatrix 17-29
- GpiSetStopDraw 13-19, 15-6
- GpiSetTag 14-3, 14-4, 14-12
 - forms
 - disk 15-1
 - memory 15-1
- GpiSetTextAlignment 6-3, 6-13, 6-14
- GpiSetViewingLimits 16-5, 16-15, 17-31, 17-32
- GpiSetViewingTransformMatrix 17-30, 17-32, 17-42
- GpiStrokePath 10-2, 10-6, 10-14
- GpiTranslate 17-19, 17-26, 17-39
- GpiUnloadFonts 1-8
- GpiUnrealizeColorTable 7-25
- GpiWCBitBit 1-8, 8-9, 8-10, 8-11, 8-15, 17-37
- GPI_ERROR 14-2
- GPI_HITS 14-2
- GPI_OK 14-2
- graphic functions
 - GpiCreatePS 9-1
 - GpiQueryFontMetrics 9-1
- graphic orders
 - adding to segments 13-5
 - code 13-4

- graphic orders (*continued*)
 - copying 13-11
 - defined for 15-2
 - definition of 13-1
 - input to GpiElement 13-4
 - memory requirements 13-2
 - sizes 13-2
- graphics
 - attributes C-1
 - resetting process C-1
 - character fonts
 - face name 9-20
 - local identifier (lcid) 9-24
 - match value 9-20
 - defining the viewing window 17-31
 - field 17-32
 - orders D-1
 - decoding D-1
 - naming conventions D-1
 - scrolling 17-32
 - viewing pipeline 17-6
 - zooming 17-32
- graphics element
 - See element
- graphics field 17-4, 17-32
- graphics field, definition 16-5
- graphics functions
 - GpiAssociate 12-11
 - GpiBeginArea 14-4
 - GpiBeginElement 13-1, 13-4, 13-5, 13-18
 - GpiBox 14-2
 - GpiCallSegmentMatrix 12-5, 12-12, 12-14
 - GpiCharString 6-13, 6-19, 6-21, 6-23, 9-14
 - GpiCharStringAt 6-13, 6-19, 6-21, 6-23, 9-14
 - GpiCharStringPos 6-13, 6-19, 6-21, 6-23
 - GpiCharStringPosAt 6-13, 6-19, 6-20, 6-21, 6-23
 - GpiCloseSegment 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-9, 13-11, 13-16, 13-18
 - GpiConvert 6-20
 - GpiCorrelateChain 14-4, 14-5, 14-11, 14-12
 - GpiCorrelateFrom 14-4, 14-12
 - GpiCorrelateSegment 14-4, 14-12
 - GpiCreateLogColorTable 7-6
 - GpiCreateLogFont 9-9, 9-20
 - GpiCreateLogicalFont 6-3
 - GpiCreatePS 12-11
 - GpiDeleteElement 13-11, 13-16, 13-18
 - GpiDeleteElementRange 13-11, 13-16, 13-18
 - GpiDeleteElementsBetweenLabels 13-11, 13-18
 - GpiDeleteSegment 13-11, 13-18
 - GpiDeleteSegments 13-11, 13-18
 - GpiDeleteSegment 13-11
 - GpiDeleteSetId 9-24, 9-25
 - GpiDestroy 12-10
 - GpiDestroyRegion 11-7
 - GpiDrawChain 12-5, 12-6, 12-7, 12-8, 12-13, 12-14, 13-7, 13-13, 13-18
 - GpiDrawDynamics 13-14, 13-16, 13-18

graphics functions (*continued*)

GpiDrawFrom 12-8, 12-14, 13-7, 13-13, 13-18
GpiDrawSegment 12-8, 12-14, 13-7, 13-13, 13-15, 13-18
GpiElement 13-4, 13-5, 13-11, 13-14, 13-18, 14-2
GpiEndArea 14-4
GpiEndElement 13-1, 13-4, 13-18
GpiEqualRegion 11-7
GpiErase 13-14, 13-18, 14-2, 14-12
GpiErrorSegmentData 13-13, 13-18
GpiFrameRegion 11-7
GpiGetData 13-4, 13-12, 13-18
GpiLabel 13-6, 13-18
GpiLine 13-2, 13-4
GpiLoadFonts 9-4, 9-14
GpiMarker 4-1
GpiMove 12-9
GpiOffsetElementPointer 13-5, 13-6, 13-16, 13-18
GpiOffsetRegion 11-7
GpiOpenSegment 12-3, 12-4, 12-11, 12-12, 12-13, 12-14, 13-1, 13-5, 13-6, 13-9, 13-10, 13-16, 13-18, 14-3
GpiPaintRegion 11-7
GpiPlayMetaFile 14-2
GpiPolyMarker 4-1
GpiPop 12-9, 14-4
GpiPtInRegion 11-7
GpiPutData 13-4, 13-10, 13-12, 13-14, 13-18, 14-2
GpiQueryAttrs 6-17
GpiQueryBackColor 6-17
GpiQueryBackMix 6-17
GpiQueryBitmapHandle 9-25
GpiQueryBoundaryData 14-12
GpiQueryCharAngle 6-8, 6-23
GpiQueryCharBox 6-7, 6-23
GpiQueryCharBreakExtra 6-15, 6-23
GpiQueryCharDirection 6-13, 6-23
GpiQueryCharExtra 6-15, 6-23
GpiQueryCharMode 6-23
GpiQueryCharSet 6-3, 6-23
GpiQueryCharShear 6-10, 6-23
GpiQueryCharStringPos 6-20, 6-21, 6-23
GpiQueryCharStringPosAt 6-20, 6-21, 6-23
GpiQueryColor 6-17
GpiQueryCp 9-9
GpiQueryCurrentPosition 6-20
GpiQueryDefAttrs 6-7
GpiQueryDefCharBox 6-23
GpiQueryDrawControl 13-18
GpiQueryDrawingMode 12-2, 13-18
GpiQueryEditMode 13-9, 13-18
GpiQueryElement 13-5, 13-6, 13-11, 13-18
GpiQueryElementPointer 13-5, 13-19
GpiQueryElementType 13-4, 13-19
GpiQueryFontFileDescriptions 9-14
GpiQueryFontMetrics 9-9, 9-23
GpiQueryFonts 9-17
GpiQueryGraphicsField 6-14, 6-20

graphics functions (*continued*)

GpiQueryInitialSegmentAttrs 12-4, 12-14, 13-7, 13-19
GpiQueryKerningPairs 9-10
GpiQueryMix 6-17
GpiQueryNumberSetIds 9-25
GpiQueryPageViewport 6-14, 6-20
GpiQueryPickAperturePosition 14-10, 14-12
GpiQueryPickApertureSize 14-10, 14-12
GpiQueryRegionBox 11-7
GpiQueryRegionRects 11-7
GpiQuerySegmentAttrs 12-14, 13-9, 13-19
GpiQuerySegmentNames 12-3, 12-14, 13-19
GpiQuerySegmentPriority 12-7, 12-14, 13-19
GpiQueryStopDraw 13-19
GpiQueryTag 14-4, 14-12
GpiQueryTextBox 6-20, 6-23
GpiQueryViewingLimit 6-14, 6-20
GpiQueryWidthTable 9-9
GpiRectInRegion 11-7
GpiRemoveDynamics 13-14, 13-16, 13-19
GpiResetBoundaryData 14-12
GpiResetPS 12-10, 12-11, 13-11
GpiRestorePS 12-10
GpiSavePS 6-7, 12-10
GpiSetAttrMode 12-9
GpiSetAttrs 6-3, 6-17
GpiSetBackColor 6-17
GpiSetBackMix 6-17
GpiSetBitmapId 9-25
GpiSetCharAngle 6-3, 6-8, 6-9, 6-23
GpiSetCharBox 6-3, 6-5, 6-7, 6-23
GpiSetCharBreakExtra 6-3, 6-15, 6-23
GpiSetCharDirection 6-3, 6-12, 6-23
GpiSetCharExtra 6-3, 6-15, 6-23
GpiSetCharMode 6-3, 6-23
GpiSetCharSet 6-3, 6-23, 9-24
GpiSetCharShear 6-3, 6-10, 6-11, 6-23
GpiSetClipRegion 11-7
GpiSetColor 6-17
GpiSetCp 9-8, 9-9
GpiSetCurrentPosition 6-21, 12-4, 12-9
GpiSetDefAttrs 6-7, 6-17, 12-4
GpiSetDefTag 14-4
GpiSetDrawControl 13-14, 13-15, 13-19, 14-2, 14-12
GpiSetDrawingMode 12-2, 12-6, 12-14, 13-2, 13-19
GpiSetEditMode 13-9, 13-16, 13-19
GpiSetElementPointer 13-5, 13-6, 13-16, 13-19
GpiSetElementPointerAtLabel 13-6, 13-16, 13-19
GpiSetInitialSegmentAttrs 12-4, 12-5, 12-12, 12-13, 12-14, 13-7, 13-9, 13-19
GpiSetMarkerSet 9-25
GpiSetMix 6-17, 13-8
GpiSetPatternSet 9-25
GpiSetPickAperture 14-11
GpiSetPickAperturePosition 14-2, 14-4, 14-12
GpiSetPickApertureSize 14-2, 14-4, 14-5, 14-12
GpiSetPS 12-11, 13-11

graphics functions (*continued*)

- GpiSetPS. 12-10
- GpiSetSegmentAttrs 12-14, 13-9, 13-19, 14-3
- GpiSetSegmentPriority 12-3, 12-7, 12-14, 13-9, 13-19
- GpiSetStopDraw 13-19
- GpiSetTag 14-3, 14-4, 14-12
- GpiSetTextAlignment 6-3, 6-13, 6-14
- GpiUnloadFonts 9-14, 9-24
- return values 14-2
- graphics interchange standard
 - Mixed Object Document Content Architecture 15-2
 - MO:DCA 15-2
- graphics objects
 - example of reflecting 17-16
 - example of rotating 17-18
 - example of scaling 17-16
 - example of shearing 17-21
 - example of translating 17-20
- graphics orders 15-2
- graphics primitives
 - color 7-1
 - description 1-3, 3-1
 - mix 7-1
 - types 1-3
- Graphics Programming Interface (GPI)
 - bar graph drawn with line and arc primitives 3-1
 - blueprint created with line and arc primitives 3-1
 - device-independence 1-1
 - introduction 1-1
 - major advantage 1-1
 - pie chart drawn with line and arc primitives 3-1
 - program execution 1-7
 - querying fonts 18-17
 - using functions 1-1
- graphics segments 1-5
 - See *also* segments
- graphics-order code
 - See element, type
- GRES_ALL 12-11, 13-11
- GRES_ATTRS 12-11
- GRES_SEGMENTS 12-11, 13-11

H

- handles
 - presentation spaces 14-5
- hard coding values for a concatenated matrix 17-25
- HCINFO 18-33
- HCINFO contents 18-12
- header file, graphics orders D-1
- healing screen with clip regions 11-8
- helper functions, transform 17-14
- helper functions, transformation 17-26
- Helvetica font 9-3
- hexadecimal identifier for graphics functions 13-2
- HIGHER_PRI 12-7

hits 14-4

homogeneous coordinate system 17-12

I

- Icon Editor 8-5
- icons
- IControl 3-10
- identifying
 - area for correlation 14-1
- identity transformation 17-1, 17-6
- image fonts 9-2
- image markers 4-3, 4-4
 - See *also* marker primitives
- image primitive 8-9
- IMAGEBUNDLE 7-25
- image, bit map 8-1
- implementing clipping 16-8
- importing
 - including
- incrementing element pointer 13-5
- index mode, color tables 7-6
- information device context 2-8
- information header, bit map 8-4
- information table, bit map 8-4, 8-16
- instance transformation 17-29
- instance, output device 2-10
- intensity, color 7-2
- interacting
- interchange requirements, metafiles 15-5
- invisible line 3-4
- invisible segments 13-8
- italic 6-1

J

- job querying and manipulation 18-31
- joining wide lines, example 10-5

K

- kernel device driver, printing 18-1, 18-4
- Kerning, graphics fonts 9-9
- keyboard
- keyname QUEUE 18-14

L

- labels
 - for locating specific elements 13-5
- label—element pair 13-5
- lcid 8-20
 - determining current value 6-3
 - of current font 6-3
 - to identify font 9-1
- lcid (logical-font identifier) 9-24, 15-15
- LCID_DEFAULT 5-4
- LCOLF_CONSECRGB 7-6

- LCOLF_INDRGB 7-6
- LCOLF_RGB 7-6
- LC_DEFAULTS 15-15
- LC_LOADDISC 15-15
- LC_NOLOAD 15-15
- leave-alone 6-16
- leave-alone mix 7-16
- LIFO (last-in-first-out) stack 12-9, 14-4
- limits, device space 17-36
- limits, world coordinate space 17-36
- limit, viewing 17-3
- line and arc functions 5-8
- line and arc primitives, drawing tools 3-1
- line color 3-2, 10-6
- line color and mix attributes 3-5
- line end 3-2
- line end attribute, description 10-4
- line end types 10-4
- line join 3-2
- line join styles 10-5
- line join, description 10-5
- line join, path attribute 10-5
- line mix 3-2, 10-6
- line primitives
 - about 3-1
 - alternate pels on 3-4
 - attributes 1-3
 - boxes 3-6
 - color and mix attributes 3-5
 - cosmetic lines 3-3
 - dash-dot line 3-4
 - dash-double-dot line 3-4
 - default values, attributes 3-2
 - description 1-3, 3-1
 - dotted line 3-4
 - double-dotted line 3-4
 - family 3-6
 - geometric lines 3-3
 - geometric width 3-2
 - invisible line 3-4
 - line color 3-2
 - line end 3-2
 - line join 3-2
 - line mix 3-2
 - line type 3-2
 - line types 3-4
 - line width 3-2
 - lines 3-6
 - long-dashed line 3-4
 - polylines 3-6, 3-7
 - series of unconnected lines 3-6
 - short-dashed line 3-4
 - standard line types 3-4
 - used for outline fonts 6-2
 - using 3-20
- line type 3-2, 10-6
- line types 3-4

- line weight
 - See fonts
- line width 3-2, 10-6
- line width for paths 10-2
- line-width attributes as interpreted by PM 3-3
- linear equations, transformation 17-11
- LINEBUNDLE 1-3, 3-2, 3-5, 3-10, 3-27, 5-9, 5-14, 5-20, 7-25, 10-2
- LINETYPE_ALTERNATE 3-4
- LINETYPE_DASHDOT 3-4, 3-10
- LINETYPE_DASHDOUBLEDOT 3-4
- LINETYPE_DOT 1-3, 3-4
- LINETYPE_INVISIBLE 3-4, 3-23
- LINETYPE_LONGDASH 3-4
- LINETYPE_SHORTDASH 3-4
- LINETYPE_SOLID 3-4
- LINEWIDTH_DEFAULT 3-3
- LINEWIDTH_NORMAL 3-3
- LINEWIDTH_THICK 3-3
- listing ports, printer drivers, queue processors 18-31
- listing printers across a network 18-31
- load balancing, printing 18-5
- loaded color tables 7-5
- local identifier
 - See lcid
- local identifier, graphics font 9-24
- locating a point in a region 11-12
- logical color table 7-2, 7-3
- logical color tables 7-3
- logical color table, creating 7-20
- logical fonts 9-17, 9-22
 - defining 9-22
- long graphics order 13-2
- long values, GpiBox 3-10
- long-dashed line 3-4
- LOWER_PRI 12-7
- LT_DEFAULT 15-14
- LT_NOMODIFY 15-14
- LT_ORIGINALVIEW 15-14

M

- major advantages of device independence 18-22
- MAKEFIXED macro 6-7, 17-14
- managing
 - print jobs 18-31
 - spooler 18-31
- manipulating
- MAP parameter 18-27
- mapping the presentation page to the device 17-34
- marker color and mix attributes 4-4
- marker primitives
 - attributes 1-4
 - color and mix attributes 4-4
 - colors 4-4
 - description 1-4
 - image 4-3, 4-4
 - vector 4-3, 4-4, 4-8, 4-10

- marker sets 9-25
- MARKERBUNDLE 1-4, 4-5, 7-25
- MARKSYM (marker symbols) 4-2
- match value, graphics fonts 9-12
- mathematics of transformations 17-11
- matrix
 - concatenated, hard coding values for 17-25
 - device transformation 17-37
 - model for building transformation matrix 17-12
 - multiplying values for 17-25
- MATRIXLF 17-13
- MATRIXLF, and reflecting 17-17
- MATRIXLF, and rotating 17-19
- MATRIXLF, and scaling 17-17
- MATRIXLF, and shearing 17-21
- MATRIXLF, and translating 17-20
- MATRIXLF, structure of 17-13
- MATRIXLF, transformation structure 17-4
- maximum baseline extent 9-12
- media space 17-4
- memory
 - management 12-10
 - memory segments differ from graphic segments 12-1
 - planning for correlation 14-10
 - requirement for graphics orders 13-2
- memory device context 2-8, 8-3
- message handling
- message queues
- messages
 - DM_PRINTOBJECT 18-26
 - WIN_REALIZEPALETTE 7-8
 - WM_BUTTON1DOWN 14-4
 - WM_PAINT 11-1
- metafile device context 2-8
- metafile handle 15-9
- metafiles
 - about 15-1
 - associating with a presentation space 15-8
 - asynchronous drawing thread restrictions 15-6
 - benefits of 15-1
 - closing 15-9
 - content restrictions 15-3
 - contents 15-1, 15-2
 - copying 15-16
 - copying to disk 15-20
 - creating 15-17
 - creating a device context 15-7
 - creating for interchange 15-5
 - deleting 15-17
 - description 1-5, 15-1
 - device context 15-7
 - disassociating from a presentation space 15-9
 - displaying contents of 15-2
 - draw-mode restrictions 15-4
 - drawing into 15-17
 - drawing into in retain mode 15-19
 - dynamic segments 15-6
- metafiles (*continued*)
 - effect of current drawing mode 15-10
 - elements 15-9
 - forms 15-1
 - graphics orders 15-2
 - handle 15-9
 - interchange requirements 15-5
 - loading from disk 15-16
 - micro presentation space restrictions 15-6
 - playing 15-2, 15-21
 - playing into a graphics presentation space 15-10
 - query restrictions 15-6
 - resetting the presentation space before playing 15-11
 - SAA-conforming 15-5
 - saving on disk 15-16
 - segment considerations 12-11
 - segment restrictions 15-6
 - sharing with other applications 15-17
 - summary of functions 15-23
 - use of the clipboard 15-17
 - using 15-17
- Metafile_NoQuery device context 2-8
- micro presentation space
 - always in draw mode 12-6
 - attribute modes invalid 12-9
 - description 2-1
 - drawing control flags 13-15
 - erase functions 13-14
 - functions for creating 2-2
 - functions valid in B-1
 - invalid for segment construction 12-3
 - no segment store 12-6
 - standard, description 2-2
- micro presentation space restrictions 15-6
- mix
 - attributes 7-15
- mix attributes 7-15
 - about 7-1
 - background 5-1, 6-16, 7-13
 - background mix 7-11
 - background mix attribute 12-8
 - background of character strings 6-17
 - character string primitives 6-16
 - description 1-4
 - for arc primitives 3-5
 - for area primitives 5-5
 - for dynamic segments 13-8
 - for line primitives 3-5
 - for marker primitives 4-4
 - foreground 5-1, 6-16
 - foreground mix 7-11
 - foreground mix attribute 12-8
 - leave-alone 6-16
 - leave-alone mix 7-16
 - no background 3-5
 - of character strings 6-17
 - OR mix 7-14

- mix attributes (*continued*)
 - overpaint 6-16, 7-14
 - specifying foreground and background 7-16
 - transparent 4-4, 5-5
 - using mix attributes 7-20
 - XOR mix 7-15
- mix attribute, definition 7-11
- Mixed Object Document Content Architecture
 - interchange standard 15-2
- modal graphic systems 2-4
- model for building the transformation matrix 17-12
- model for building transformation matrix 17-12
- model space 1-6, 17-3
- model space-to-page space transformations 17-30
- model transformation 17-3, 17-28
- modification operations, path 10-13
- modify path 10-9
- modifying
 - job parameters 18-31
- monochrome devices, color graphics 7-18
- monospace font 6-4
- moving
 - dynamic segments 13-14
 - element pointer 13-5
 - regions 11-6
 - through coordinate spaces 17-6
 - to next page 18-23
- MO:DCA interchange standard 15-2
- MPATH_STROKE 10-13
- multiple arcs 3-10
- multiple queues 18-5
- multiple-arc primitive family 3-17
- multiplying matrix values 17-25

N

- naming
 - segments 12-3
 - segments in consecutive order 12-3
 - unique names for unchained segments 12-5
- negative character chear 6-11
- network printing considerations 18-26
- non-modal graphics systems 2-5
- non-retained graphics
 - conversion to retained graphics 12-11
 - correlation on 14-2
 - creating 12-11, 14-2
 - description 1-5
 - detectable 14-2
 - disadvantages compared to retained graphics 12-1
 - drawing output 12-1
 - identifiers for correlation 14-2
 - nonretained segments 12-2, 12-11
 - preparing for correlation 14-1
- nondelectable segments 13-8
- nonretained segment 12-11, 13-9
- nonretained-drawing mode 17-3

- normal device context, description 2-8
- normal presentation space
 - attribute modes valid 12-9
 - description 2-4
 - drawing control flags 13-15
 - erase functions 13-14
 - memory considerations 12-10
 - valid for segment construction 12-3
 - WinBeginPaint 2-4
- notification codes
- notification messages

O

- obtaining
 - cached micro presentation spaces, table 2-3
 - presentation spaces 2-6
- OD_MEMORY 8-3
- OD_METAFILE 15-7
- OD_METAFILE_NOQUERY 15-6, 15-7
- OD_QUEUED 18-11
- offsetting a region 11-11
- 1-byte graphics order 13-2
- open segments 13-9, 13-11, 13-13
- operations
- operations, bit map 8-3
 - creating
 - with an application 8-3
- operations, region 11-4
- OR mix 7-14
- order
 - See graphics order
- orders, graphics D-1
- outline fonts 9-2
 - Adobe fonts 9-4
 - average character width 9-13
 - code pages 9-8
 - face name 9-5
 - family name 9-5
 - installing 9-14
 - kerning 9-9, 9-13
 - maximum baseline extent 9-12
 - modes 9-13
 - private 9-14
 - proportional spacing 9-9
 - public 9-14
 - simulation 9-12
 - system-provided 9-2
- outline path 10-9
- output page, creating 18-23
- overpaint 6-16
- overpaint mix 7-14

P

- page coordinate space 1-6
- page setup dialog, description 18-9

- page space 17-3
- page space-to-device space transformations 17-33
- page viewport 17-33, 17-34
- page viewports 17-34
- painting
 - region 11-12
- palette manager 7-8
- parallel port driver, printing 18-4
- parameter values
 - BBO_AND 8-12
 - BBO_IGNORE 8-12
 - BBO_OR 8-12
 - BM_DEFAULT 7-13, 15-5
 - BM_LEAVEALONE 7-13, 7-19, 15-5
 - BM_OR 7-13
 - BM_OVERPAINT 7-13, 15-5
 - BM_XOR 7-13
 - CAPS_ADDITIONAL_GRAPHICS 7-17
 - CAPS_RASTER_CAPS 8-15
 - CBM_INIT 8-4, 8-6
 - CLR_BACKGROUND 7-4, 7-5, 7-18, 8-9
 - CLR_BLACK 7-4
 - CLR_BLUE 7-4
 - CLR_BROWN 7-4
 - CLR_CYAN 7-4
 - CLR_DARKBLUE 7-4
 - CLR_DARKCYAN 7-4
 - CLR_DARKGRAY 7-4
 - CLR_DARKGREEN 7-4
 - CLR_DARKPINK 7-4
 - CLR_DARKRED 7-4
 - CLR_DEFAULT 7-4, 7-5, 7-18
 - CLR_FALSE 7-4
 - CLR_GREEN 7-4
 - CLR_NEUTRAL 7-4, 7-5, 7-18
 - CLR_PALEGRAY 7-4
 - CLR_PINK 7-4
 - CLR_RED 7-4
 - CLR_TRUE 7-4
 - CLR_WHITE 7-4, 7-18
 - CLR_YELLOW 7-4
 - CRGN_AND 16-7
 - CRGN_DIFF 16-7
 - DBM_HALFTONE 8-9
 - DBM_INVERT 8-9
 - DBM_STRETCH 8-9
 - DCTL_DISPLAY 15-8
 - DCTL_OFF 15-8
 - DM_DRAW 8-9, 8-10, 15-3, 15-10
 - DM_DRAWANDRETAIN 8-10, 15-3, 15-5, 15-10
 - DM_RETAIN 8-10, 15-3, 15-5, 15-6, 15-10
 - FM_AND 7-12
 - FM_DEFAULT 7-12, 15-5
 - FM_INVERT 7-12
 - FM_LEAVEALONE 7-12, 15-5
 - FM_MASKSRCNOT 7-12
 - FM_MERGENOTSRC 7-13
 - FM_MERGESRCNOT 7-12

- parameter values (*continued*)
 - FM_NOTCOPYSRC 7-13
 - FM_NOTMASKSRC 7-13
 - FM_NOTMERGESRC 7-12
 - FM_NOTXORSRC 7-12
 - FM_ONE 7-13
 - FM_OR 7-12, 15-5
 - FM_OVERPAINT 7-12, 15-5
 - FM_SUBTRACT 7-12
 - FM_XOR 7-12
 - FM_ZERO 7-12
 - LCOLF_CONSECRGB 7-6
 - LCOLF_INDRGB 7-6
 - LCOLF_RGB 7-6
 - OD_MEMORY 8-3
 - OD_METAFILE 15-7
 - OD_METAFILE_NOQUERY 15-6, 15-7
 - PATSYM_DEFAULT 7-18
 - PATSYM_SOLID 7-18
 - PMF_LOADTYPE 15-5
 - ROP_DSTINVERT 8-14
 - ROP_GRAY 8-15
 - ROP_MERGECOPY 8-14
 - ROP_MERGEPAINT 8-14
 - ROP_NOTSRCCOPY 8-14
 - ROP_NOTSRCERASE 8-14
 - ROP_ONE 8-14
 - ROP_PATCOPY 8-14
 - ROP_PATINVERT 8-14
 - ROP_PATPAINT 8-14
 - ROP_SRCAND 8-14
 - ROP_SRCRCOPY 8-14
 - ROP_SRCERASE 8-14
 - ROP_SRCINVERT 8-14
 - ROP_SRCPAINT 8-14
 - ROP_ZERO 8-14
 - SCP_ALTERNATE 16-4
 - SCP_AND 16-3
 - SCP_RESET 16-3
 - SCP_WINDING 16-4
 - SYSCLR_ACTIVEBORDER 7-10
 - SYSCLR_WINDOW 7-18
- parameters
 - ARE 18-27
 - COL 18-27
 - configuration 18-6
 - COP 18-27
 - DEVESC_NEWFRAME 18-23
 - DevOpenDC 18-19
 - DPDM_QUERYJOBPROPS 18-15
 - FIT 18-27
 - for correlation 14-5
 - GpiFullArc input 3-14
 - IMaxDepth (correlation) 14-8
 - IMaxHits (correlation) 14-5
 - IType (correlation) 14-5
 - MAP 18-27
 - pcTotal 18-13

- parameters (*continued*)
 - PMPRINT/PMPLLOT queue processor 18-27
 - using ARE and FIT 18-29
 - XFM 18-27
- PARCPARAMS 3-10
- partial arcs 3-12, 3-14
- partial arc, example 3-16
- path color and mix attributes 10-6
- path identifier 10-8
- paths 17-38
 - alternate and winding fill modes 10-11
 - area attributes 10-6
 - attribute default values 10-2
 - attributes 10-2
 - brackets 10-8
 - clip 17-3
 - clipping 10-15
 - color and mix attributes 10-6
 - constraints for stroking 10-14
 - constructed in different directions in winding mode 10-12
 - constructed in same direction in winding mode 10-12
 - conversion to clip path 10-15
 - conversion to region operation 10-16
 - cosmetic line attribute 10-6
 - creating hollow characters 10-10
 - creating triangular clip path 10-19
 - description 1-5, 10-1
 - determining filled portion 10-11
 - drawing filled polygons 10-17
 - drawing geometric (wide) line 10-16
 - drawing outline text 10-18
 - drawing techniques 10-1
 - fill operations 10-11
 - FPATH_ALTERNATE 10-16
 - FPATH_WINDING 10-16
 - functional sequence for drawing outline 10-10
 - functions that generate 10-9
 - functions used within path bracket 10-9
 - functions valid in B-1
 - geometric width 10-2
 - line width 10-2
 - list of operations 10-9
 - modification operations 10-13
 - operations 10-9
 - stroking operations 10-14
 - tasks 10-16
 - using 10-16
 - winding mode 10-12
- PATSYM_DEFAULT 5-3, 7-18
- PATSYM_SOLID 5-3, 7-18
- PATSYN_ERROR 5-3
- pattern reference point 5-1, 11-3
- pattern reference point attribute, description 5-3
- pattern set 5-1, 11-3
- pattern set attribute, description 5-4
- pattern sets 9-25
- pattern symbol 5-1, 11-3
- pattern symbol attribute 5-2
- pcTotal 18-13
- pDriverData 18-14
- PD_JOB_PROPERTY flag 18-26
- pel 7-1
- pels 6-8, 6-16, 8-1
- phosphor 7-1
- physical color table 7-3, 7-7
- physical fonts 9-17
- pick apertures 14-1, 14-2, 14-11
- pick identifiers 14-4
- pick tags 14-4
- PICKSEL_VISIBLE 14-11
- picture elements 8-1
- pictures, shearing 17-41
- pictures, translating, rotating, and scaling 17-39
- PICVIEW application, using 18-2
- pie chart drawn with line and arc primitives 3-1
- pie slice, drawing 3-23
- pixels 8-1
- PMF_COLORREALIZABLE 15-11
- PMF_COLORTABLES 15-11, 15-12
- PMF_DEFAULTS 15-11, 15-12
- PMF_LCID 15-12
- PMF_LCIDS 15-11
- PMF_LOADTYPE 15-5, 15-11, 15-12
- PMF_RESET 15-11
- PMF_RESOLVE 15-11
- PMF_SEGBASE 15-11
- PMF_SUPPRESS 15-11, 15-12
- PMORD.H, graphics orders header file D-1
- PMPRINT/PMPLLOT queue processor
 - parameters 18-27
- PM_Q_RAW 18-2
- PM_Q_STD 18-2
- PM_Q_STD format, advantages 18-2
- PM_Q_STD format, content restrictions 18-2
- PM_SPOOLER application name 18-14
- point size 6-1, 6-4, 6-8, 6-19, 6-22
- pointer to graphics element 13-5
- POINTL 5-20, 6-23, 11-12
- points
 - definition of 9-1
- POLYGON 5-13, 5-20
- polygon boundaries 5-14
- polygon construction 5-14
- polygon primitives
 - about 5-13
 - description 5-13
 - POLYGON_BOUNDARY 5-14
 - POLYGON_NOBOUNDARY 5-14
- polygons
 - construction 5-14
 - description 1-3
 - overlap 5-15
 - primitive, description 1-3

- printer setup dialog 18-13
- printer setup dialog, description 18-9
- printer sharing, description 18-5
- PRINTERINFO 18-33
- printer's points
 - See points
- printing
 - aligning text 6-20
 - application design considerations 18-22
 - bypassing GPI presentation layer 18-30
 - data flow 18-6
 - drag/drop protocol considerations 18-26
 - dynamic segments 13-14
 - formatting text on a page 6-19
 - forms selection 18-11
 - job submission and manipulation 18-1
 - kernel device driver 18-1, 18-4
 - load balancing 18-5
 - network, considerations 18-26
 - non-PM jobs 18-9
 - on a separate thread 18-19
 - overview of application interface and data flow 18-7
 - PM_Q_STD to PM_Q_RAW conversion 18-2
 - print job formats 18-2
 - printer setup dialog 18-13
 - printer-specific format, creating 18-3
 - queued PM print job 18-9
 - user interface 18-1
- printing to a file, methods 18-25
- prioritizing print jobs 18-1
- priority of segments 12-3, 12-8
- private fonts 9-14
- PRJINFO2 18-33
- PRJINFO3 18-33
- processing
 - print jobs 18-18
- process, definition 1-7
- program execution 1-7
- propagate detectability attribute of graphics segments 13-8
- propagate visibility attribute of graphics segments 13-8
- proportional font 6-4
- proportional spacing, graphics fonts 9-9
- PRPORTINFO 18-33
- PRPORTINFO1 18-33
- PRQINFO3 18-33
- PRQINFO3 structure 18-13, 18-14
- PRQINFO6 18-33
- PRQPROCINFO 18-33
- pszComment 18-13
- pszPrinters field 18-15
- PS_NORESET 12-11
- public fonts 9-14
- PU_LOENGLISH 3-22

Q

- QMOPENSTRUC 18-33
- quarter-circle box-corner rounding 3-9
- query functions, list 5-9
- query restrictions 15-6
- querying
 - default queue 18-14
 - device 18-18
 - device fonts 18-17
 - devices 18-23
 - invalid in retain drawing mode 12-2
 - list of available printers 18-13
 - valid in draw-and-retain drawing mode 12-3
- queue creation, querying, and manipulation 18-31
- queue description 18-6
- queue drivers, printing 18-1
- queue driver, description 18-4
- queue manipulation 18-4
- queue processors, printing 18-1
- queue processor, description 18-4
- queued device context 2-8
- queued printing, description 18-7

R

- raster fonts 9-2
- raster output device 8-1
- raster-operation (ROP) value 8-13
- reading
- reassociating presentation space with device contexts, example 18-21
- RECTL 11-2, 11-7, 11-9, 11-13, 16-12, 16-15
- redrawing windows
 - using bit maps 8-2
- reference segment 12-7
- reflecting a graphics object, example 17-16
- reflecting, and MATRIXLF 17-17
- reflection transformation 17-15, 17-16
- region handle 11-1
- regions
 - attributes 11-2
 - changing definition 11-5
 - combining 11-3, 11-10
 - combining disjoint 11-5
 - comparing 11-11
 - complex, definition 11-5
 - converting to clip region 11-7
 - coordinates 11-3
 - creating a region 11-4
 - creating and deleting 11-9
 - creation 11-3
 - defining 11-1
 - defining a region, example 11-2
 - deletion 11-9
 - description 1-5, 11-1
 - destination 11-4
 - determining characteristics 11-6

regions (continued)

- determining coordinates of rectangles 11-13
- functions to use with current clip 11-8
- handle 11-3
- healing screen with clip regions, example 11-8
- locating a point 11-12
- moving 11-6
- offsetting 11-11
- operations 11-4
- painting 11-12
- purpose 11-2
- source 11-4
- target 11-4
- using 11-9
- ways to combine 11-5

regions, path conversion to 10-16

relationship between queues and devices 18-5

releasing

renaming segments 13-12

requesting

requirements for direct spooling 18-30

resources

responding

responding to application queries, printing 18-4

restoring

- presentation space 2-6

RES_DEFAULT 15-12

RES_NORESET 15-12

RES_RESET 15-12

retain 12-2

retained graphics

- about 12-1
- advantages 1-5
- advantages over nonretained graphics 12-1
- conversion from nonretained graphics 12-11
- correlating after drawing 14-1
- correlation on 14-3
- creating 12-12
- description 1-5
- disadvantages 1-5
- drawing 12-6, 12-12
- editing 13-1
- graphics segments 1-5
- storing output 12-1
- summary of drawing functions 12-14

retained-drawing mode 17-3

retrieving

- job properties 18-14
- rectangles 11-7

return values of functions 14-2

RGB 7-25, 8-7

RGB color encoding 7-2

RGB mode, color tables 7-6

RGB values 7-2

RGB2 7-25, 8-7, 8-24

RGNRECT 11-13

root segments 12-5, 12-7, 12-8, 13-7, 13-13, 14-8

root segments, applying transformations to 17-24

ROP_DSTINVERT 8-14

ROP_GRAY 8-15

ROP_MERGECOPY 8-14

ROP_MERGEPAINT 8-14

ROP_NOTSRCCOPY 8-10, 8-14

ROP_NOTSRCERASE 8-14

ROP_ONE 8-14

ROP_PATCOPY 8-14

ROP_PATINVERT 8-14

ROP_PATPAINT 8-14

ROP_SRCAND 8-14

ROP_SRCCOPY 8-14

ROP_SRCERASE 8-14

ROP_SRCINVERT 8-14

ROP_SRCPAINT 8-14

ROP_ZERO 8-14

rotating a graphics object, example 17-18

rotating, and MATRIXLF 17-19

rotating, scaling, and translating a picture 17-39

rotation transformation 17-17

round-off error 17-27

routes for printer data 18-7

rubber-banding a straight line, sample code 3-20

rubber-banding effect, straight line 3-20

S

sample code

- combining regions 11-10
- creating a non-display device context for a printer 2-9
- creating and deleting a region 11-10
- creating custom fill pattern from a font character 5-19
- creating custom fill pattern from bit map 5-16
- creating standard device context for metafile 2-9
- defining an area primitive 5-7
- determining rectangle coordinates 11-14
- DEVOPENSTRUC 2-9
- displaying job properties dialog 18-15
- drawing a closed figure 5-15
- drawing a fillet 3-25
- drawing a hyperbolic curve 3-25
- drawing a pie slice 3-24
- drawing a spline 3-26
- drawing a straight line 3-20
- drawing an ellipse 3-23
- drawing filled polygons 10-17
- drawing geometric (wide) line 10-16
- drawing multiple intersecting closed figures 5-16
- drawing outline text 10-18
- GpiEqualRegion 11-11
- locating a point in a region 11-12
- offsetting a region 11-11
- painting a region 11-12
- querying the print using DevQueryHardcopyCaps 18-11

sample code (*continued*)

- rubber-banding a straight line 3-20
- saving
 - presentation space 2-6
- scale-down transformations 17-3
- scaling a graphics object, example 17-16
- scaling transformation 17-15
- scaling, and MATRIXLF 17-17
- scaling, translating, and rotating a picture 17-39
- scan line 8-1
- SCP_ALTERNATE 16-4
- SCP_AND 16-3
- SCP_RESET 16-3
- SCP_WINDING 16-4
- scrolling 17-32
- searching
- SEGE_M_INSERT 13-4, 13-10, 13-12
- SEGE_M_REPLACE 13-4, 13-10
- segment restrictions 15-6
- segment transformation 17-29
- segments
 - adding elements 13-4
 - adding to segment chain 12-4
 - advantages of unchained segments 12-5
 - all called segments visible 13-7
 - attributes
 - about 12-4
 - altering for individual segments 13-9
 - applying to primitives outside of 13-9
 - ATTR_CHAINED 13-7
 - ATTR_DETECTABLE 13-7, 14-3
 - ATTR_DYNAMIC 13-7, 14-3
 - ATTR_FASTCHAIN 13-7
 - ATTR_PROP_DETECTABLE 13-7, 14-3
 - ATTR_PROP_VISIBLE 13-7
 - ATTR_VISIBLE 13-7
 - called 13-8
 - chain 13-13
 - chain OFF
 - chained 12-4, 12-5, 12-12, 12-13
 - changing 13-8
 - detectable 13-8, 14-2
 - dynamic 13-7, 13-14
 - dynamic OFF
 - fast-chained 12-4, 12-5, 12-11
 - initial 12-4, 13-6, 13-7
 - introduction to 13-6
 - invisible 13-8
 - nondetectable 13-8
 - nondynamic 13-9
 - ON and OFF settings 13-6
 - open 13-13
 - propagate-detectability 13-8
 - propagate-visibility 13-8
 - unreliable after closing 12-4
 - visible 13-8
 - visible OFF
- ATTR_CHAINED 12-4, 12-5

segments (*continued*)

- ATTR_FASTCHAINED 12-4, 12-5, 12-11
- bracket 13-1
- called 17-24, 17-29
- called from other segments 17-29
- called segments 12-5, 12-8, 12-9, 14-8
- chain 12-13, 13-11, 14-4, 14-5
- chained segments 12-8
- closing 12-4
- contents 12-4
- converting chained to unchained 13-9
- converting dynamic to nondynamic 13-14
- correlation 14-4
- correlation for chained segments 14-4
- correlation hits for called segments 14-8
- creating 12-3
- creating called segments 12-13
- creating chained segments 12-12
- creating retained graphics 12-1
- deleting 13-11
- deleting a range 13-11
- deleting from segment store 13-11
- detectable 14-3
- different from memory segments 12-1
- drawing 12-6, 13-13, 13-14
- drawing dynamic 17-29
- drawing entire chain 13-13
- drawing errors
 - GPIE_DATA 13-14
 - GPIE_ELEMENT 13-14
 - GPIE_SEGMENT 13-14
- drawing mode when created 13-2
- drawing on an output device 13-13
- drawing order 13-13
- drawing partial chain 13-13
- drawing retained graphics 12-1
- drawing segment chains 12-13
- dynamic
 - editing 13-1, 13-9
 - element pointer value 13-5
 - element pointers 14-4
 - elements 13-3, 14-4
 - establishing priority 12-7
 - functions valid in B-1
 - GpiBeginElement 13-4
 - GpiEndElement 13-4
 - GpiSetDrawingMode 13-2
 - higher priority 12-7
 - identifiers for correlation 14-3
 - in consecutive order 12-3
 - in hardcopy output 13-14
 - in normal presentation spaces 12-3
 - inheriting attributes 13-8
 - initial 12-12, 12-13
 - inserting data in 13-9
 - invisible 14-3
 - labels 13-5
 - label—element pair 13-5

segments (*continued*)

- lower priority 12-7
- metafile considerations 12-11
- naming 12-3
- nondynamic 14-3
- nonretained 13-9, 14-2
- nonretained segments 12-2, 12-11
- open 13-9, 13-11
- priority 12-3, 12-8, 14-8
- propagated detectable 14-3
- reference segment 12-7
- renaming 13-12
- replacing data in 13-9
- replaying for correlation 14-1
- root segments 12-5, 12-7, 12-8, 13-7, 13-13, 14-8
- selectable 13-8
- source 13-12
- status 12-6, 13-2
- storing GPI functions 13-1
- storing graphics orders 13-2
- structure of 13-1
- tagging primitives 14-4
- unchained 12-5, 13-7
- undetectable for correlation 14-4
- zero segments 12-3, 12-5, 13-9, 13-11

selecting

- area for correlation 14-1

sending

serial port driver, printing 18-4

serif 6-1

- See also fonts
- definition 9-1

session, definition 1-7

setting

- current position 3-2
- drawing units 17-39

sharpness of a fillet 3-18

shearing a graphics object, example 17-21

shearing pictures 17-41

shearing transformation 17-21

shearing, and MATRIXLF 17-21

short-dashed line 3-4

simple drawing applications 3-1

simple-arc primitive family 3-12

SIZEL 2-14

sizing

- pick apertures 14-2, 14-11

soft fonts, description 18-17

source region 11-4

source segment 13-12

space characters 6-15

specification functions, list 5-8

specifying

- color output 18-27
- colors for monochrome print 18-27
- number of copies 18-27
- printing area 18-27

- SplControlDevice 18-32
- SplCopyJob 18-32
- SplCreateDevice 18-31, 18-32
- SplCreateQueue 18-31, 18-32
- SplDeleteDevice 18-32
- SplDeleteJob 18-32
- SplDeleteQueue 18-32
- SplEnumDevice 18-32
- SplEnumDriver 18-32
- SplEnumJob 18-32
- SplEnumPort 18-32
- SplEnumPrinter 18-31, 18-32
- SplEnumQueue 18-13, 18-32
- SplEnumQueueProcessor 18-32
- SplEnumyyy 18-31
- SplHoldJob 18-32
- SplHoldQueue 18-32
- spline with no discontinuity of gradient 3-19
- splines 3-10, 3-17
- spline, drawing 3-26
- spline, example 3-19
- SplPurgeQueue 18-32
- SplQmAbort 18-32
- SplQmAbortDoc 18-32
- SplQmClose 18-32
- SplQmEndDoc 18-32
- SplQmOpen 18-32
- SplQmStartDoc 18-32
- SplQmWrite 18-32
- SplQueryDevice 18-32
- SplQueryJob 18-32
- SplQueryQueue 18-14, 18-32
- SplReleaseJob 18-32
- SplReleaseQueue 18-32
- SplSetDevice 18-32
- SplSetJob 18-32
- SplSetQueue 18-32
- SplxxxDevice 18-31
- SplxxxJob 18-31
- SplxxxQueue 18-31
- spool files, description 18-2
- spooler 18-1
- spooler functions
- standard bit-map formats 8-7
- standard line ends 10-4
- standard micro presentation space 2-2
- starting
 - document printing 18-18
 - print job 18-22
- static appearance, bit maps 8-5
- stroke path 10-9
- stroke width 6-1
- stroking operations, path 10-14
- structures
 - ARCPARAMS 3-10
 - AREABUNDLE 3-10, 5-9, 5-13, 7-25, 10-2
 - BITMAPINFOHEADER2 8-4, 8-6
 - BITMAPINFO2 8-6

structures (continued)

- BUNDLE 7-1
- CHARBUNDLE 6-1, 6-2, 6-3, 6-17, 6-23, 7-25
- DEVOPENSTRUC 2-9, 2-14, 15-8, 18-33
- DRAGITEM 18-26
- DRIVDATA 18-14, 18-33
- DRIVPROPS 18-33
- FATTRS 6-1, 6-23
- FONTMETRICS 6-1, 6-5, 6-23
- HCINFO 18-12, 18-33
- IMAGEBUNDLE 7-25
- LINEBUNDLE 3-2, 5-9, 5-14, 7-25, 10-2
- MARKERBUNDLE 7-25
- PARCPARAMS 3-10
- POINTL 6-10, 6-23, 11-12
- POLYGON 5-13
- PRDDRIVINFO 18-33
- PRDINFO3 18-33
- PRINTERINFO 18-33
- PRJINFO2 18-33
- PRJINFO3 18-33
- PRPORTINFO 18-33
- PRPORTINFO1 18-33
- PRQINFO3 18-13, 18-33
- PRQINFO6 18-33
- PRQPROCINFO 18-33
- QMOPENSTRUC 18-33
- RECTL 11-2, 11-7, 11-13, 16-12, 16-15
- RGB 7-25
- RGB2 7-25
- RGNRECT 11-13
- SIZEL 2-14
- summary of line and arc 3-27

styles

subpictures 17-1

- definition of 13-2

summary

- area primitive structures 5-20
- line and arc primitive functions 3-27
- line and arc primitive structures 3-27
- of correlation functions 14-12
- of editing segment functions 13-18
- of retained graphics drawing functions 12-14
- OS/2 area functions 5-20
- OS/2 bit-map functions 8-24
- OS/2 bit-map structures 8-24
- OS/2 clipping functions 16-15
- OS/2 clipping structure 16-15
- OS/2 color and mix attribute functions 7-24
- OS/2 color and mix attribute structures 7-24
- OS/2 metafile functions 15-23
- path functions, table 10-20
- path structures, table 10-20
- presentation space 2-6
- presentation space and device context functions 2-13
- presentation space and device context structures 2-13, 2-14

summary (continued)

- printing and spooler functions 18-32
- printing and spooler structures 18-33
- region functions 11-15
- region structures 11-15
- transformation functions, table 17-44
- transformation structures, table 17-44, 17-45

support

- SUP_NOSUPPRESS 15-13
- SUP_SUPPRESS 15-13
- Symbol font 9-3
- symbol point code, definition 5-2
- SYSCLR_ACTIVEBORDER 7-10
- SYSCLR_WINDOW 7-18
- system colors 7-9
- system font 9-2
- system fonts, description 18-17
- system implementation, region 11-2

T

table

- actual drawing mode 12-6
- area attribute default values 5-2
- background mix attributes 7-13
- base pattern set 5-2
- bit map functions 8-24
- bit map structures 8-24
- character modes 6-4
- character string primitive data structures 6-23
- character string primitive functions 6-23
- character string primitive initial defaults 6-3
- clipping areas and coordinate space
 - summary. 16-2
- common form sizes 18-11
- comparison of paths and areas 10-1
- default logical color table 7-4
- direction of an arc 3-11
- drawing mode dependencies when playing metafiles 15-10
- effect of drawing mode on segments 13-3
- eight standard colors 7-2
- fillet sharpness values 3-18
- foreground mix attributes 7-11
- function sequence for drawing outlines 10-10
- function to create normal presentation space 2-4
- functions for creating micro presentation spaces 2-2
 - functions that draw multiple arcs 3-17
 - functions that draw straight lines 3-6
 - functions that obtain cached micro presentation spaces 2-3
- how fonts obey character attributes 6-4
- initial segment attributes 13-7
- line attribute default values 3-2
- line width attributes as interpreted by PM 3-3
- marker primitive initial defaults 4-2
- mix value names 8-14

table (continued)

- normal device contexts 2-8
- of drawing control flags 13-15
- of graphics orders 13-2
- presentation space and device context functions, summary 2-13
- presentation space and device context structures 2-14
- presentation space and device context structures, summary 2-13
- presentation space features and restrictions 2-6
- printing and spooler structures, summary 18-33
- rgb values 7-2
- rules for implementing clipping 16-8
- standard bit-map formats 8-7
- standard geometric line end types 10-4
- standard geometric line join types 10-5
- standard line types 3-4
- summary of area functions 5-20
- summary of area primitive structures 5-20
- summary of correlation functions 14-12
- summary of editing segment functions 13-19
- summary of line and arc functions 3-27
- summary of line and arc primitive structures 3-27
- summary of path functions 10-20
- summary of path structures 10-20
- summary of printing and spooler functions 18-32
- summary of region functions 11-15
- summary of region structures 11-15
- summary of retained graphics drawing functions 12-14
- summary of transformation functions 17-44
- summary of transformation structures 17-44, 17-45
- Types of Coordinate Spaces 1-6
- tags for correlation 14-3
- target
- target presentation space 8-15
- target region 11-4
- terminating
- text alignment 6-13, 6-14, 6-20
- text formatting 6-19
- thread, definition 1-7
- tilted ellipse 3-14
- Times New Roman font 9-3
- transformation matrix 3-11
- transformations
 - about 17-4
 - and coordinate spaces 17-1
 - applying 17-7
 - applying transformation functions 17-27
 - applying transformation operations directly to the transformation matrix 17-27
 - coding the device transformation 17-36
 - combining between a coordinate space pair 17-10
 - concatenating 17-24
 - current 17-22
 - current transformation value 17-22
 - default device 17-34, 17-36

transformations (continued)

- default viewing 17-32
- device 17-33, 17-34, 17-36, 17-37
- equations 17-11
- existing 17-27
- functions 1-6, 17-14
- functions and types 17-4
- functions, about 17-22
- helper functions 17-14, 17-26
- identity 17-1, 17-6
- instance 17-29
- linear equations 17-11
- mathematics 17-11
- MATRIXLF 17-13
- model 17-3, 17-28
- model for building matrix 17-12
- model space-to-page space 17-30
- of bit-map data 17-37
- operations 1-6
- operations, about 17-14
- other than the identity transformation 17-7
- page space-to-device space 17-33
- reflection 17-15, 17-16
- rotation 17-17
- scale-down 17-3
- scaling 17-15
- segment 17-29
- shearing 17-21
- translation 17-19
- unity 17-6
- viewing 17-30, 17-42
- windowing-system 17-37
- world space-to-model space 17-30
- world to model space transformations 17-41
- world-to-model-space 17-27
- transformations and coordinate spaces 17-1
- transformations and coordinate spaces, using 17-39
- transforming
- transforming bit-map data 17-37
- TRANSFORM_ADD 17-24
- translating a graphics object, example 17-20
- translating, and MATRIXLF 17-20
- translating, rotating, and scaling a picture 17-39
- translation transformation 17-19
- triangular clip path, creating 10-19
- triangular clip path, example 10-15
- true WYSIWYG 18-22
- 2-byte graphics order 13-2
- type of graphics elements 13-4
- types of device contexts 2-7

U

- unchained segment 15-10
- unchained segments 12-5, 13-7, 13-9
- understanding
- unit circle 3-10

- unity transformation 17-6
- user interface, printing 18-1
- using
 - ARE and FIT parameters 18-29
 - area primitives 5-15
 - bit maps 1-4
 - DevEscape with DEVEESC_ABORTDOC 18-23
 - DevEscape with DEVEESC_STARTDOC 18-22
 - device contexts 2-9
 - device fonts 18-17
 - graphics functions 1-1
 - job properties dialog 18-14
 - line and arc primitives 3-20
 - page setup dialog 18-10
 - paths 10-16
 - PICVIEW 18-2
 - presentation spaces 2-6
 - print dialog and print processing 18-18
 - region functions 11-9
 - SplEnumQueue 18-13
 - spooler queue 18-1
 - system fonts 18-17
- using coordinate spaces and transformations 17-39
- using world to model space transformations 17-41

V

- variables
- variations of straight and curved lines 3-2
- vector font 9-2
- vector markers 4-3, 4-4, 4-8, 4-10
 - See also marker primitives
- very long graphics order 13-2
- viewing
 - PM_Q_STD-format jobs 18-2
- viewing limit 17-3
- viewing pipeline 16-2, 17-2, 17-6
- viewing transformation 17-42
- viewing transformations 17-30
- viewing window 17-31
- viewing window, defining 17-31
- viewing window, definition 16-5
- viewport, page 17-34
- views
- visibility
- visibility attribute of graphics segments 13-8

W

- ways to combine regions 11-5
- width vectors 6-15
- WinBeginPaint 2-2, 2-3, 2-4, 2-6, 2-13
- winding mode, description 5-11
- winding mode, paths in 10-12
- window device contexts, description 2-7
- window functions
 - WinDrawText 9-14
 - WinGetPS 9-1

- window functions (*continued*)
 - WinQueryCpList 9-9
 - WinQueryUpdateRegion 11-8
- windowing-system transformation 17-37
- WinDrawBitmap 8-9, 8-24
- WinEndPaint 2-6, 2-13
- WinGetPS 2-3, 2-6, 2-13
- WinGetScreenPs 2-2, 2-3, 2-13
- WinOpenWindowDC 2-7, 2-11, 2-13
- WinQuerySysColor 7-9
- WinQuerySysValue 2-13
- WinQueryUpdateRegion 11-8
- WinQueryWindowDC 2-11, 2-13
- WinRealizePalette 7-8
- WinReleasePS 2-6, 2-13
- WIN_REALIZEPALETTE 7-8
- WM_BUTTON1DOWN 14-4
- WM_PAINT 11-1
- working
- world coordinate space 1-6, 17-1, 17-2
- world coordinate space position 2-5
- world coordinate space, limits 17-36
- world coordinates
 - break values 6-15
 - calculating width of output 6-20
 - character cell dimensions 6-7
 - used for character cells 6-5
- world space-to-model space transformation 17-30
- world-to-model-space transformations 17-27
- writing
 - segment contents 12-4

X

- XFM parameter 18-27
- XOR mix 7-15
- XOR mode 13-8
- XOR raster operation 13-7

Z

- zero element position 13-5
- zero segments 12-3, 12-5, 13-8, 13-9, 13-11
- zooming 17-32

Numerics

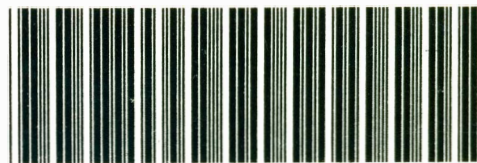
- 3-point arcs 3-12, 3-16

® IBM, OS/2 and Operating System/2 are
registered trademarks of
International Business Machines Corporation



© IBM Corp. 1992
International Business
Machines Corporation

Printed in the
United States of America
All Rights Reserved
10G6495



S10G-6495-00



P10G6495